University of Ljubljana
Faculty of Computer and Information Science

Barbara Krašovec

Distributed Computing as a Service with ARC middleware

Master's thesis

Mojca Ciglarič
advisor

Andrej Filipčič
coadvisor

Ljubljana, 2016

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Barbara Krašovec

Porazdeljeno računalništvo kot storitev z uporabo vmesne programske opreme ARC

MAGISTRSKA NALOGA

Mojca Ciglarič
MENTOR

Andrej Filipčič
SOMENTOR

Ljubljana, 2016

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Večna pot 113
1000 Ljubljana, Slovenija
telefon: 01 47 98 100
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Številka: 144-MAG-ISO/2016
Datum: 29. 02. 2016

Barbara KRAŠOVEC, prof. franc. in univ. dipl. pol.

Ljubljana

Fakulteta za računalništvo in informatiko Univerze v Ljubljani izdaja naslednjo magistrsko nalogo

Naslov naloge: **Porazdeljeno računalništvo kot storitev z uporabo vmesne programske opreme ARC**

Distributed computing as a Service with ARC middleware

Tematika naloge:

Integracija grida in oblaka je v znanstvenem svetu zelo aktualna tema. Grid se je pojavil v 90. letih in je hitro postal široko uporabljena arhitektura za izvedbo računskih nalog in različnih znanstvenih projektov. Njegova uporaba je zelo kompleksna, arhitektura sama ima nekaj težje premostljivih omejitev. Prva so računski viri, ki so omejeni in večkrat polno zasedeni, druga so relativno omejena okolja za izvajanje nalog. Oblak bi v tem okviru dobro izpopolnil pomanjkljivosti grida, saj bi v primeru večjih obremenitev in zasedenosti kapacitet omogočal razširitev arhitekture grid v oblak, hkrati bi z virtualiziranimi sistemi omogočil uporabniku izvedbo nalog v kateremkoli okolju, ki bi ga za svoje delo potreboval. Čeprav sta arhitekturi v svojih ciljih in sestavnih komponentah precej podobni, sta konceptualno različni, zato se pri njuni integraciji soočamo s številnimi izzivi. Kako arhitekturi združiti, da bo vseeno ohranjen visok nivo varnosti in zanesljivosti, bo tema te magistrske naloge. Za združevaje arhitektur bomo uporabili vmesno programsko opremo ARC.

Mentorica:
doc. dr. Mojca Ciglarič

Fakulteta za računalništvo in informatiko 2 UNIVERZA V LJUBLJANI

Dekan:
prof. dr. Nikolaj Zimic

Somentor:
prof. dr. Andrej Filipčič

IZJAVA O AVTORSTVU MAGISTRSKEGA DELA

Spodaj podpisana Barbara Krašovec, z vpisno številko 63080478, sem avtorica magistrskega dela z naslovom:

PORAZDELJENO RAČUNALNIŠTVO KOT STORITEV Z UPORABO VMESNE PROGRAMSKE OPREME ARC

in angleškim naslovom

DISTRIBUTED COMPUTING AS A SERVICE WITH ARC MIDDLEWARE

S svojim podpisom zagotavljam, da:

- sem magistrsko delo izdelala samostojno pod vodstvom mentorice doc. dr. Mojce Ciglarič in somentorstvom prof. dr. Andreja Filipčiča,
- so elektronska oblika magistrskega dela, naslov (slov., angl.), povzetek (slov., angl.) in ključne besede (slov., angl.) identični s tiskano obliko magistrskega dela,
- soglašam z javno objavo elektronske oblike magistrskega dela v zbirki "Dela FRI".

V Ljubljani, dne 31. avgusta 2016 Podpis avtorice:

ACKNOWLEDGEMENTS

Firstly, I would like to thank my co-advisor dr. Andrej Filipčič for sharing his knowledge, for all his advice and help. Without his support this dissertation would not have been possible. Furthermore, I sincerely thank my advisor dr. Mojca Ciglarič and dr. Farid Ould-Saada for their comments and help. Last but not least, I would like to thank my colleagues at Arnes and Jozef Stefan Institute and especially to my family and friends for their support.

— Barbara Krašovec, Ljubljana, September 2016.

CONTENTS

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Barbara Krašovec
Porazdeljeno računalništvo kot storitev z uporabo vmesne programske opreme ARC

POVZETEK

Postopki znanstvene računalniške obdelave so se v zadnjih letih hitro razvijali. Znanstvene programske rešitve so postajale vedno bolj kompleksne in zahtevajo za svojo obdelavo veliko količino računskih kapacitet. Močno je narasla uporaba arhitekture grid, ki je za uporabo precej zahtevna, a vendar jo za izvedbo svojih projektov in eksperimentov uporablja ogromno raziskovalcev. Računske in diskovne kapacitete so v gridu omejene, heterogene in pogosto polno zasedene. Heterogena ni le strojna oprema, temveč tudi programska oprema. Nastavitve in spremembe konfiguracije programske opreme v gridu so omejene, kar otežuje prilagajanja izvedbenih okolj in enostavno migracijo aplikacij iz grida v druge računalniške arhitekture. Uporaba arhitekture oblaka bi lahko v znanstvenem svetu odpravila veliko težav. V oblaku so kapacitete na voljo na zahtevo, lahko jih poljubno dodajamo in krčimo. Prav tako arhitektura omogoča več svobode pri izbiri programske opreme in nastavitvah le-te v oblačnih instancah.

Zanimanje za združevanje arhitekture grida in oblaka je veliko, saj bi nova infrastruktura lahko omogočila povsem enostavno migracijo aplikacij iz grida v oblak ali obratno. Poraba računskih virov bi bila optimalna, zagotovljena bi bila razširljiva, zanesljiva in trajnostna arhitektura. Na področju oblakov, bi lahko Advanced Resource Connector (ARC) odigral ključno povezovalno vlogo, vlogo razpršenega visokozmogljivega računanja kot storitve. ARC omogoča dostop do razpršenih računskih virov, se razvija v skladu z odprtimi standardi in v smeri, ki bi omogočala lažje združevanje z drugimi arhitekturam. Tudi v hibridni arhitekturi bi ARC omogočil raziskovalcem, da do različnih razpršenih virov dostopajo na že poznan in uveljavljen način, ki ga uporabljajo na gručah grid.

Nekaj modelov združevanja arhitektur že obstaja, a ponavadi rešujejo le problematiko točno določene znanstvene discipline. V tem delu bomo predstavili nov model združevanja grida in oblaka, ki bo uporabnikom omogočil uporabo dodatnih računskih kapacitet v zasebnih in javnih oblakih. Za dostop do dodatnih kapacitet bo uporabil od-

jemalec ARC, ne da bi bil primoran spremeniti svoje izvedbene programe, kodo ali sam eksperiment. Gre za globalno rešitev, ki omogoča postavitev gruče grid v kateremkoli zasebnem ali javnem oblaku, ter ustreza definiciji samodejne razširitve oblaka (ang. cloud bursting).

Ključne besede: grid, oblak, računalništvo v oblaku, znanstvena računalniška obdelava, porazdeljeno računalništvo, združevanje tehnologij, visokozmogljivo računanje, razširitev oblaka

University *of Ljubljana*
Faculty *of Computer and Information Science*
Barbara Krašovec
Distributed Computing as a Service with ARC middleware

ABSTRACT

Scientific computing has evolved considerably in the past years. Scientific applications became more complex and require an increasing number of computing resources to perform well on a large scale. Grid computing became widely used and is the chosen infrastructure for many scientific calculations and projects, although it demands a high learning curve. Computing and storage resources in grid are limited, heterogeneous and often overloaded. Heterogeneity is not present only in the hardware setups, but also in software composition, where configuration permissions are limited. This heterogeneity hardens the portability of scientific applications. Usage of cloud resources could eliminate those constraints. In cloud, resources are provisioned on demand and can scale up and down, scientists can easily customize their execution environments in the form of virtualization.

The interest for grid and cloud integration is big, since the new infrastructure would enable a relatively simple migration of scientific applications from grid to cloud (or vice versa). The utilization of resources would be more efficient and it would provide scalability, sustainability and reliability of the hybrid infrastructure. In the IaaS domain, Advanced resource connector (ARC) middleware could have a role of distributed high throughput batch computing as a service. ARC has an ability to facilitate distributed computing across organizations. It is strongly committed to open standards and interoperability and in a good position to enable users and research communities to access new resources and new platforms using existing and well-established interfaces and standards. It could provide a way to seamlessly evolve its interfaces to enable existing users and communities to take full advantage of new resources and technologies as they become available.

Some integration models exist, but usually only correspond to a specific use case. In this thesis we introduce a new model of grid and cloud integration which enables users to

benefit from additional private and public cloud resources via ARC, with no modifications of the code, workload or execution scripts on their side, by setting up a virtual grid cluster on demand in automated way. The solution can be globally used, on all private and public clouds, it is scalable and corresponds to the cloud bursting paradigm.

Key words: grid, cloud, interoperability, integration, scientific computing, distributed computing, technology integration, high performance computing, cloud bursting

I

Introduction

1.1 Motivation

Distributed computing has evolved considerably in the past years. Distribution is present in many different infrastructures, such as clusters, grids [1], clouds [2] [3], storage systems, high performance computing (HPC), networks etc. Their complexity is linked to different devices used, different policies, different hardware, formats, protocols and network connections [4]. While distributed systems evolved, so did the applications that run on them. High-speed networks, infiniband technology and communication protocols have reduced the bandwidth and latency gaps among dispersed nodes and processors that are present in a distributed system [5]. Grid computing was introduced in the 1990s, based on the same idea as the Web. The main purpose of this technology was to share not only information stored on a distributed computer, but also computing and storage resources. Resources are available to the user on demand [6]. Distribution in grid can be seen as a distribution of resources and a distribution of computational jobs. The distribution of resources covers administration and management of resources. Distributed resources are main components for job submission. As for jobs in the distributed cluster, jobs are coupled on nodes that are in the same cluster. Despite the complexity of grid, this architecture is widely used for many scientific computations and experiments. Computing power requests are increasing, grid clusters are usually overloaded, while they provide limited physical resources and they can only provide restricted software customizations and virtual organizations (VO) support [7]. They cannot always respond to high user demands. Applications need to be tested in each grid environment and adjusted accordingly, user tasks are queued and executed when their priority is high enough. In peak times conventional resources do not suffice, user communities would like to benefit from the resources in private or public clouds.

Cloud-based solutions are being considered as a new approach to scientific computing, whereas cloud offers seemingly infinite resources and is easily accessible to all users. It provides on-demand resource provisioning, enables support for multiple execution environments, outsources maintenance costs for an organization and is simple to use. The availability of resources in the cloud is dynamically growing and decreasing by demand, therefore resources are available for a limited period of time. Grid on the other hand has limited resources that are available on a long-term basis, although the amount of resources is not constant [8] and job processing time is relatively short. However, sci-

entific projects and applications still rely on a relatively permanent High Performance Computing (HPC) [9] and High Throughput Computing (HTC) [10] [11] resources, while clouds in this context can be regarded as a "spill-over" resource that can be used during peak times when conventional facilities are overloaded or when prompt results are needed (when a deadline of a project is approaching) – this is the so-called Cloud Bursting [12]. Optional addition of cloud-based resources to grid infrastructure is an important step towards enhancing traditional scientific computing.

Many different distributed models have been introduced that combine or integrate two different technologies, such as grid, web, parallel or cloud components. Those changes require new approaches in system administration, system performance tuning, maintenance, usage, methodologies etc [13].

Many scientific initiatives are aiming for extending grid with some external cloud resources, especially in peak times when local resources are overloaded, with none or minimal management effort. For now no common solution is available. Approaches to the integration vary. Most user communities adjust their workload to be executed in the public cloud, which is costly and time consuming, due to nonstandardized API-s and software in public clouds, users are bound to a single commercial provider. Some solutions include building scientific private clouds with some grid components, such as OpenStack [14] private cloud with grid authentication and authorization VOMS [15]. Other available solutions join grid and cloud by developing interoperable user interfaces that enable job execution on different architectures.

Why different approaches? Several problems are emerging when running scientific applications in the public cloud. To name a few, public clouds have proprietary application programming interfaces (API), different image formats, different network connection possibilities, different contextualization of the cloud, accounting, not to mention data management challenges and its integration with the cloud storage. Initiatives solve those problems partially, in different ways or they avoid them by some workarounds, which is why the approaches are so different. They all have some advantages, but their common disadvantage is that they only benefit for a specific research community or a specific scientific project or they are a single commercial cloud compatible.

After studying the available solutions and describing the general problems of grid and cloud integration, we identified the main objectives of our new hybrid architecture:

- possibility to extend grid cluster with cloud resources in times of peak demand,

- authentication and authorization mechanisms adopted from grid computing, even though they are complex, they reassure a certain level of security and privacy and they enable a collaboration of users among multiple domains,
- automated resource provisioning,
- optimal resource utilization,
- scientific workload can run on both architectures with no modifications,
- virtual instances are added automatically with no user intervention,
- portability of scientific applications from grid to any private or public cloud.

1.2 Scientific Contributions

The main and original contributions of this work are:

- thorough analysis of deployment options, of technical and functional requirements for grid and cloud integration,
- evaluation of available grid and cloud integration attempts and solutions, design of a hybrid manager that will enable managing, assigning and controlling of grid and cloud resources,
- grid and cloud integration design that enables usage of cloud resources when grid resources are unavailable or used, design of a new hybrid distributed architecture, of an elastic cluster based on ARC middleware. The objective is to provide a hybrid system with grid authentication model and high security level that enables registration of services, jobs, users, service discovery, job execution, reservations, distributed workload and scalability.

1.3 Methodology

The objectives of this work are designing and deploying a hybrid distributed architecture that would integrate grid and IaaS cloud architectures using ARC middleware [52] [18] [53]. Approaches to cloud and grid integration vary, therefore many different solutions

are available and they usually only focus on a certain workload and cannot be applicable in other fields or to other types of workloads. Our work is divided into the following phases:

1. ANALYSIS OF TECHNOLOGIES USED: First we analyzed ARC middleware and its constituent services, functionalities, protocols and standards. Further on we analyzed local resource managers and identified the possibilities for cloud bursting implementation. We analyzed the suggested and available grid-cloud integration solutions and cloud stacks and their features and utilization possibilities.

2. CONCEPTUALIZATION: We defined an appropriate number of abstract classes – integration models to enable classification of suggested and available integration solutions and architectures. We classified all the integration solutions according to the abstract classes.

3. EVALUATION OF AVAILABLE INTEGRATION MODELS: We evaluated the above mentioned integration models in detail.

4. PILOT TESTBED: We deployed a pilot environment for job execution, setting up a grid cluster using ARC middleware and local HTCondor manager [16]. We deployed a hybrid cloud, based on OpenStack and EC2 [17].

5. INTEGRATION OF GRID AND CLOUD: We suggested a new integration model or a new setup, which will enable job execution in the cloud via grid middleware using existing workload descriptions and will not require any modifications on the user side. We adjusted local resource manager, authentication and authorization mechanisms used in grid, to be used in the hybrid infrastructure. Information services were adapted.

6. IMPLEMENTATION: We implemented the suggested model as a proof of concept and we identified its eventual bottlenecks and problems.

7. EVALUATION OF THE HYBRID ARCHITECTURE: We evaluated the performance and stability of the new hybrid architecture. We identified bottlenecks and possible overhead in the new architecture and proposed some optimization possibilities.

1.4 Research domain overview

Grid computing was introducted in the 1990s, when the main challenge in scientific computing was to integrate multiple dispersed computing centers into a single system in such a way that users would not be troubled with where the data is, how to authenticate, if they are even authorized to use the system, which nodes are available etc. The idea was that each center would be independently managed and available to several different disciplines and users at the same time [18]. Information technology specialists at CERN [19] were searching for an appropriate toolkit for Large Hadron Collider grid (WLCG) [20]. A few prototypes were developed and tested, taking Globus Toolkit as the base. In 2003 Data Grid project [21] started which focused on Globus [22] and Condor interoperability [23]. The following EGEE I, II and III projects from 2004 to 2010 [21] were focusing on the gLite middleware [24] and its integration with different Local LRMS systems [25]. The goal of those project was to attract new disciplines to use gLite on grid clusters. In 2010 European Grid Initiative (EGI) [21] was born, which helped the countries to take over the management in funding of the national grid infrastructure under their domain. European Middleware Initiative [26] was founded and collected all the grid software into the common repositories and coordinated the development of different middlewares and their integration with all available batch systems.

In the past few years, a lot of interest and development is directed to the usage of private and public clouds for scientific workload and to grid and cloud integration. Clouds offer a seemingly infinite computing and storage resources that represent an attractive addition to the grid. Moreover, clouds meet various end-user demands and are considered as a new approach to scientific computing. Grid and cloud are conceptually similar, both technologies make use of distributed resources, both are scalable and both use some kind of scheduling software. But the objective of their usage is clearly different. Grids are used for job executions and are typically used for short-term programs, computations and scrips, while clouds focus on long-term services. Clouds rely on virtualization of the hardware, grids have heterogeneous physical hardware behind a batch system.

Joining grid and cloud resources in a new infrastructure includes much more than just additional resources. It provides sustainability, enables complementary usage of resources, efficient resource utilization, load balancing, automatic resource management and dynamic provisioning. It also solves one of the main drawbacks of grid – in grid, new applications usually need to be tested in multiple different environments, where user has

limited configuration permissions and limited resources available. Cloud enables deploying an entire cluster. Furthermore the concept of grid and cloud integration is applicable in many different ways, depending on the interpretation of both architectures, middleware, software used and problems that it is solving.

This thesis is organized as follows. In Chapter 2 grid and cloud architectures will be described and compared. In Chapter 3 we will discuss the advantages of each architecture, common properties, disadvantages. We will identify the key factors for grid and cloud integration and describe them. In Chapter 4 we will present the benefits of the new hybrid infrastructure and the main difficulties we have to surpass when building it. Interoperability and integration will be discussed in Chapter 5. In Chapter 6 we will describe our testbed infrastructure, our grid cluster and private cloud. Possible grid and cloud integration models will be described and evaluated in Chapter 7. In Chapter 8 we will comment our setup and explain our own integration model and discuss results in Chapter 9.

2

Distributed Computing

Distributed computing is evolving, systems are becoming bigger and more complex. The term distributed computing is not new, as it appeared already before the invention of Internet; the predecessor of Internet, ARPANET, offered a distributed application already in 1970s – email [27]. Distribution is present in many different infrastructures, in clusters, grids, clouds, storage systems, HPC clusters, networks, databases, user management etc [1]. Distributed systems entail a great number of heterogeneous and dynamic physical resources. With the evolution of the systems, applications evolved simultaneously, so did resource management, maintenance, usage. The changes relate to multiple dimensions, such as [13]:

- level of concurrency in applications,
- dynamic and scalable data sets,
- decentralization of control and management of resources, data, applications,
- communication protocols,
- parallel programming efficiency.

The main advantage of the distributed system is its resilience, the failure of one component in the system does not influence the availability of the system as a whole. Grid and Cloud are both distributed infrastructures. In grid computing power and storage is shared among multiple clusters and locations, in cloud, tasks or jobs are not tightly coupled but highly distributed and independent of the cluster itself.

Already in 1991, Peter Deutsch, who worked as a programmer on distributed systems, published 7 fallacies about the distributed computing [28], eighth was added by James Gossling in 1997:

- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- The network is secure.
- Topology doesn't change.

- There is one administrator.
- Transport cost is zero.
- The network is homogeneous.

Those statements were commented years later (e.g. [29]), but authors realized, all the facts are still true, they are only more obvious than two decades ago. The key to developing a good distributed application is a broad understanding of the network. All the fallacies need to be acknowledged, otherwise problems will arise sooner or later.

There are, however, a lot of advantages using distributed systems compared to the centralized setup.

- horizontal scalability for applications,
- elimination of a single point of failure,
- better price/performance,
- high reliability and availability,
- load distribution,
- resource sharing,
- flexibility,
- more total power.

Main disadvantages of distributed systems are linked to the required low latency and high network throughput, which is not easily accomplished. With the access to multiple systems, security mechanisms have to be implemented too. Adding to that it is complex to develop software that can run on a distributed system.

2.1 Distribution of Computing

High performance computing (HPC) is used for parallel tasks and applications. A single server in the cluster is not treated autonomously, but is part of a large supercomputer. One task runs simultaneously on multiple processors within multiple servers. This is so called horizonal scalability of an application [30]. To achieve high performance computing, multiple conditions must be fulfilled [5]:

- fast inter-process connections that include high bandwidth and low latency (e.g. using infiniband technology to connect multiple servers)
- fast hardware (CPU-s)
- scalable software that fully utilizes the resources on dispersed nodes.

Grid covers multiple types of distribution: distribution of resources, distribution of tasks that run on those resources, distribution of system administration and operational tasks, distribution of components for job submission and distributed user management. Cloud in this aspect includes distributed resources, their management and services that run on them, user management is centralized.

3

Grid and Cloud Architectures

3.1 Grid Computing

Grid was born out of the needs in high-energy physics and is now used in several scientific disciplines, such as engineering, business and research. It enables access to large computing centers. Grid takes its name from electric power grid where accessing computing power would be as simple as accessing electric power. Both, electric and computing grid, provide an efficient networked infrastructure, general power in large installation systems and they both enable simple integration of new distributed providers [1]. In 1998 Carl Kesselman and Ian Foster proposed a simple definition of grid [31]: "A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities." But grid in not only a set of heterogeneous hardware and software resources, it consists also of virtual organization management, application software for virtual organization support, authentication and authorization mechanisms, access to distributed data, information services, name services etc. A few years later, Ian Foster has published a "Grid checklist" in 2002 [32][32] in which he defines grid as a system that: "coordinates resources that are not subject to centralized control, uses standard, open, general-purpose protocols and interfaces, delivers nontrivial qualities of service." Not all distributed systems are grid [33]. Grid enables access to limited group of users (members of virtual organizations), their access to resources is therefore controlled and limited by certain rules, on the other hand grid reassures a certain level of quality of service and security. Grid harnesses distributed heterogeneous resources and enables on-demand provisioning. Services are shared between organizations, computer power and data are shared over the Internet. Grid enables execution of large-scale simulations and computations, it uses data that is available in distributed databases and therefore requires a reliable and stable network to access them. There is no grid without a sustainable and fast network. In grid researchers are organized in virtual organizations (VO-s) that demand a specific virtual research environment (VRE) for their computations and simulations. VO membership determines user role and permissions within the VO and on the clusters [34].

Grid is suitable for the execution of different types of tasks, such as distributed tasks, high performance tasks, high througput tasks, on-demand tasks, data-intensive tasks and collaborative tasks [35]. The way they are used depends on the use case.

3.1.1 From Clusters to Grids

In Europe, CERN [36]has played a crucial role in grid computing development. The construction of Large Haddron Collider (LHC) [37] accelerator was approved in 1996 which led to different studies of how many computer resources will be required for the experiment. The numbers were barely describable, but it was obvious that CERN will not be capable of providing such resources on its own, which is why the idea of distributed grid centers was presented as a solution. In 1998, with the project MONARC [38], LHC network design has been prepared with the objective to solve the problem of multiple computing centers integration in such way that users would see a single service/system, not being concerned about where the data is, which resources to use, which are even available or how they should authenticate to use them. LHC network was divided into Tier-1 and Tier-2 centers. CERN as Tier-0 was intended for initial data processing and master copies archive, Tier-1 centers for high throughput jobs and storage capabilities, Tier-2 centers were designed as centers for end-user analysis and simulations [20]. Each center would be managed independently and would support different research disciplines that are running computational jobs on the cluster. Many efforts have been made in finding a suitable software for cluster management on dispersed locations. In USA, Foster and Kasselman have introduced I-Way [39] [40], ancestor of today's Globus Toolkit, as their grid middleware solution. In Italy INFN institutes have tested Condor (today HTCondor) [25] and managed to connect their institutes with it, while CERN was testing different possibilities for a few years, tried customizations of Globus, then in 2003, with the project Open Data Grid [38], LHC grid computing was finally deployed, using customized Globus Toolkit and Condor (in the USA the Open Science Grid project [1] was in place at the same time). Through the projects of EGEE I,II and III, gLite middleware was developed and is still used today. In the Nordic countries groups of developers and scientists started to develop their own software in the Nordugrid project, Advanced Resource Connector (ARC) . Globus Toolkit, Nordugrid ARC, gLite and Unicore [41] are still used in the European grids and develop in order to use the latest technologies and to respond to the needs of various research communities. After EGEE projects, EGI-Inspire project [42] started, aiming for inviting new research communities to the grid. User portals were developed to facilitate the grid submission process and to adjust runtime environments. Software and middleware development was unified and standardized to some extent, under the aegis of EMI middleware project

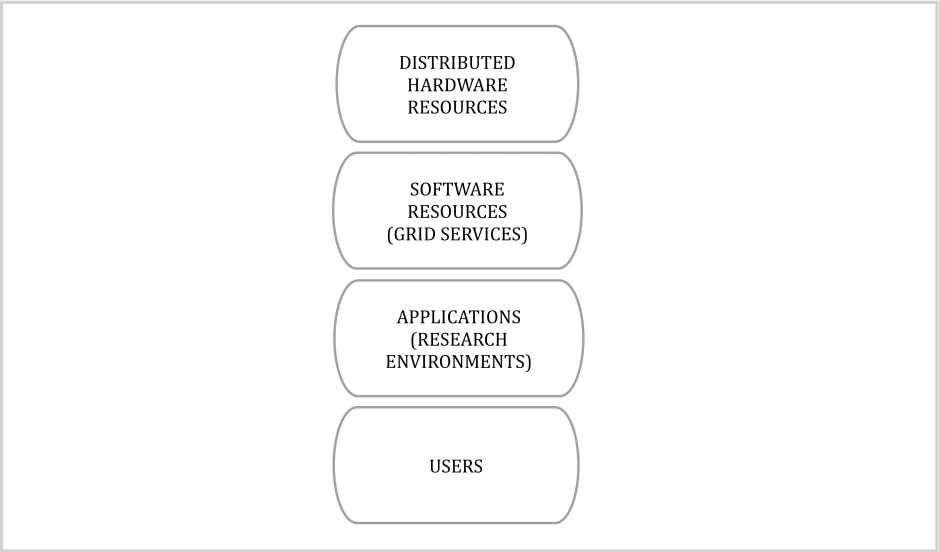


Figure 3.1
The structure of grid computing

[26]. In order to protect themselves from frequent changes, CERN started to build their own distributed framework with their own interfaces to the grid, such as PanDA, Alien and Dirac [33].

3.1.2 Grid Structure

The main components of grid computing are distributed hardware resources with networking, grid middleware, applications that run on the middleware and users that use the applications (see Fig. 3.1). A user submits a job from her home domain, requires an application and resources. A resource broker (part of the grid middleware) selects the domain, appropriate resources and executes the task.

3.1.3 Main Grid Components

Grid computing is a juncture of distributed software and hardware resources and consists of the following main components:

- job submission management with security framework (authentication, authorization, access to the infrastructure),
- resource brokering (scheduling),
- data transfer and data management,

- resource allocation,

which are all included in the grid middleware. Grid middleware is a connector between hardware resources, software resources and research environments of the VO-s. It consists of a user interface that enables job submission, job retrieval, data transfer, of monitoring services, accounting services, security mechanisms and virtual organization management. It enables access to data, configurations on the cluster, temporary job results and job status. Highly used middlewares are gLite, Nordugrid ARC [43], in USA OSG [33] and Unicore [44].

Definition 1: Grid middleware is a software stack, a junction between operating system and applications that hides heterogeneous resources and makes them appear as homogeneous. The software includes information about resources, registration of jobs and services, execution, resource monitoring, failure detection, storage and standardized access to the clusters.

Definition 2: Interware is a software framework for distributed computing. It provides an interface to access multiple distributed resources, such as grid and cloud.

An important component is meta-scheduling or scheduling, as some authors call the resource broker. It is a software that includes a policy, which job can be executed on a certain resource and when. Common schedulers in grid computing are SLURM [45], PBS [46], OGS [47], HTCondor [16]. Scheduling differs from cluster to cluster, some prioritize large jobs that demand a lot of resources and backfill the small jobs until the resources are drained, some prioritize small jobs, ”first came” jobs (FIFO scheduling), jobs that are related (related tasks run simultaneously on different processors - GANG scheduling) or smallest comulative demand first [3]. Scheduling is a very complex component of grid and requires a lot of knowledge of the middleware, hardware, software and tasks that run on the cluster. Well-configured methods for scheduling tasks benefit the utilization of the cluster as a whole and job execution speed. Apart from the middleware, a reliable and fast network is required and a dependable file transfer protocol for grid to function well. Data is transferred to the grid using a user interface or by accessing an external storage node. When transferred from external nodes, data is cached and results transferred back to the storage node, when the jobs are finished.

Definition 3: Meta-scheduler, scheduler, batch system, distributed resource manager (DRM), distributed resource management system (DRMS), workload automation or resources broker are all synonyms for a software which manages and controls background services for job execution in grid, clusters and HPC.

3.2 Cloud Computing

Cloud computing is a system, defined by three main characteristics: virtualization, service oriented architecture and on-demand resource provisioning. Cloud is a quickly scalable, dynamically configured, distributed and virtualized system [48], where resources are available on demand and payed as you go [6].

Cloud computing provides a business model that provides on-demand provisioning of virtual resources as metered services [3]. There are various definitions of cloud computing. Vaquero et al. [49] suggest 20 definitions of cloud and conclude that "Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization". Most authors concur that basic cloud computing characteristics include on-demand services, scalable and distributed resources and databases, virtualization and ease-of-access to remote sites [50] [51]. Some definitions do not provide a differentiation of an IaaS cloud and grid or other distributed technologies. I. Foster et al. [6] define cloud as a distributed system which is identified by the following characteristics:

- scalability,
- dynamic configuration of services,
- delivery of different level of services for users outside of cloud,
- availability on demand.

IBM defines cloud as services and data in scalable data centers that can be accessed from anywhere by any device [52]. From all those definitions we can conclude that every virtualization cannot be characterized as cloud. Clouds are automated, on-demand services and orchestration that do not require any manual configuration or settings.

Definition 4: Clouds are automated, scalable, on-demand services and orchestration that correspond to various use cases, can be configured to adjust variable load and allow optimal resource utilization.

3.2.1 Cloud Variants

Cloud distinguishes between three variants, three deployment models:public (also called community or commercial cloud), private and hybrid cloud.

- Public clouds offer seemingly infinite resources to external users by the policy of pay-as-you-use, resources are dynamically growing and decreasing by demand. The availability of resources is therefore limited to a certain period of time. Public cloud is available to the general public under the same conditions, it is managed and owned by a cloud provider, such as DigitalOcean [53], Amazon EC2 [54], Google Compute Engine [55], Rackspace [56]. Cloud providers offer instances that differ in capacities and performance.
- private cloud can be built using different platforms [57]: OpenNebula [58], Eucalyptus [59], OpenStack [60], oVirt [61] etc.
- hybrid cloud is a composition of clouds, usually of private and public cloud. Data and applications are portable, but clouds remain separate entities [48].

In the past years numerous academic cloud implementations emerged, attracted by its flexibility and cost reduction possibility. Those clouds tend to focus on research environments and application provisioning for specific research discipline, ignoring the computation performance, data transfers and network obstacles in the first line.

3.2.2 Cloud Service Models

There are three service models of the cloud [62] [63] [64]:

- Cloud Software as a Service (SaaS) is a model where the user uses an application in the cloud that is predefined by the cloud provider. Network, operating system, storage and software are chosen and set by the cloud provider. The installation, configuration, deployment and maintenance are all done automatically. The example of such a service is Gmail.

- Cloud Platform as a Service (PaaS) is a model where the user uses network, operating system and storage, predefined by the cloud provider and installs its own software on the platform. Usually multiple platforms are available to be chosen from. The installation, configuration, deployment and maintenance are all done automatically. The example of such a model is Google Cloud Engine.

- Cloud Infrastructure as a Service (IaaS): IaaS provides access to raw computing resources and software, end-user can run his own operating system, his own images. Network and storage are set by the cloud provider, a user can install desired operating system and software. The installation, configuration, deployment and maintenance are all done automatically. The example of such a service model is Amazon EC2.

This work focuses entirely on the IaaS cloud, while it provides access to raw computing resources. A user can build its own operating system and software on top of the virtualized environment. IaaS cloud offers more flexibility than PaaS or SaaS cloud, since the installed software is not predefined, and is suitable for deploying legacy code. IaaS cloud reassures less overhead than PaaS and SaaS cloud, whereas it can adapt to different user demands and can be used more efficiently. Cloud instances are provisioned using a user-defined image and are released when they are not needed anymore.

3.2.3 Cloud and Virtualization

Equating virtualzation to the cloud is wrong. Virtualization is part of the cloud, it enables the cloud. Cloud is an automated virtualization, an automatic creation of services that demands no human intervention.
Virtualization refers to the creation of a virtual version of some technology, there are many types of virtualization, such as storage virtualization, file virtualization, network virtualization or CPU virtualization. Virtualization contains a virtual layer above the phyisical layer, a hypervisor, which enables the separation between application view of resources and infrastructure itself [33]. When using virtualization, virtual instances coexist on the same system and share hardware. Each instance uses its own operating system. The performance and utilization of one instance can effect the performance of another one, not due virtualization itself, but due to lower ratio of CPU cache, CPU interrupts that come from other running applications, network i/o operations and multiple accesses to the disk [30].

Table 3.1

Grid and Cloud Comparison

Property	GRID	CLOUD
Ownership of resources	by public institutions	by cloud providers/private companies
Type of resources	Persistent	On-demand
Resource availability	Highly available	Available on demand
Interactive interface	No	Yes
Customized environment	Partially (RTE)	Yes
Federation of resources	Accross multiple domains	Single domain
API-based access	No	Yes
Maintenance costs	Yes	No
Security of access	PKI,x509,AAI,SSL	HTTPS,SSO,x509
Business model	SLA	SLA, billing, accounting
Failure management	Limited (failed jobs are restarted)	Replication and failover support
User management	Decentralized, VO-based	Centralized
Workload management	Built-in	None

3.3 Grid and Cloud comparison

Grid and cloud architectures have a lot in common, both are based on distributed resources, both are scalable, both enable access to large computing and storage resources [65]. Cloud relies entirely on virtualization of the hardware while grid has heterogeneous physical hardware behind a batch system. An important difference is that grids are typically used for job execution, a program execution of limited duration, often part of a larger set of jobs, and producing altogether a significant amount of data [66], while cloud supports a job usage pattern, they seem more often used to support long-lasting services. Cloud is service-oriented, grid is service and application oriented [6]. Orchestration is different in grid and cloud, it is impossible to run jobs or services in the same way on different infrastructures (see Tab. 3.1).

The main objective of grid is to provide a unified layer on top of heterogeneous hardware, which comprises a common authorization, a common job and user management,

a common job descriptions (task specifications) etc. All this is included in a grid computing element (CE), which communicates with grid clients. Apropos of grid services, grid environment consists of job submission, resource management, scheduling, data management and transfer, resource allocation and user authorization and policies, which are all included in the grid middleware [67]. Grid's constituent services include, apart from compute elements, storage elements, virtual organization management system (VOMS) [7], information system and security framework. On top of those services, a reliable file transfer protocol is available, distributed databases and workload management system.

Cloud computing provides on-demand provisioning of virtual resources as metered services [3]. It is a scalable, flexible infrastructure that complies to different user demands and enables easy access to resources from any device. It comprises an image storage, a block storage, an authorization service, an object storage, a network service and compute nodes.

Both architectures have their advantages. Grid supports running multiple tasks simultaneously, enables multiple users to run their computations on the same physical infrastructure and ensures a relatively high security level with a well-defined authentication and authorization mechanisms. Grid is resilient, a failure of one grid centre does not influence the availability of the grid as a whole. Its heterogeneous hardware and software are transparent to the user. On the other hand, it is quite complex to use, has limited resources and can guarantee only partial customizations of the execution environments for the VO-s.

Cloud is simple to use, very flexible, it easily scales and is very reliable. Resources are not owned like in grid, but leased when needed, which nullifies maintenance costs and over-provisioning. The price for this flexibility and simplicity are low security standards and proprietary software. Due to nonstandardized software, public cloud providers are locked-in and do not enable any interoperability with other public cloud solutions. The usage of cloud is intended for long-term services, not short-term data intensive computations, which means that data and network traffic are not properly addressed and demand high usage costs. In our hybrid architecture, we should focus on the positive aspects of both architectures and use them as the basis of our integration model (see Fig. 3.2). Grid with federated resources is multi-task and multi-tenant, a suitable architecture for versatile applications and VO-s, it has a well established authorization procedure via VOMS and delegated credentials. Cloud is flexible and provides a simple resource provisioning via API, web interface or some other software, enables deployment of various execution

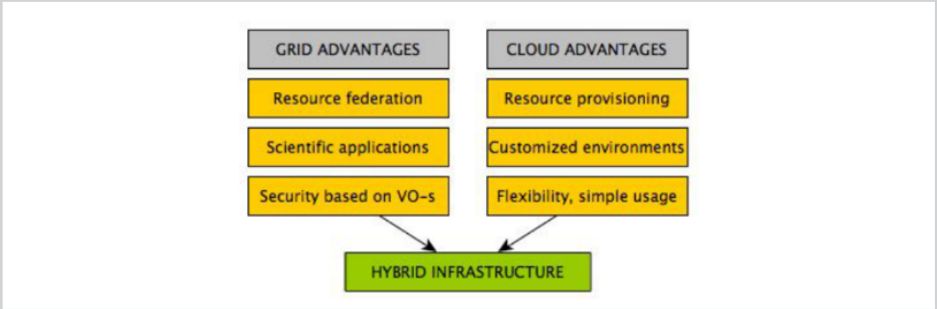


Figure 3.2
Advantages of Grid and Cloud Architecture

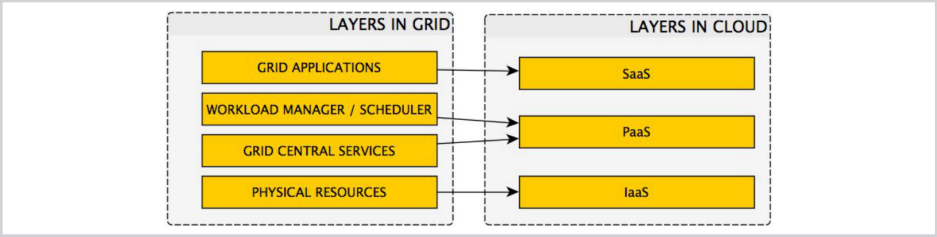


Figure 3.3
Layers of grid and cloud

environments and is easily accessible to all users.

Fig. 3.3 shows that grid services belong to the Platform as a Service (PaaS) layer [48] [52] since they provide interfaces to resources. In a PaaS cloud, scheduling is transparent to the user and is done in the background, whereas in grid the user has control over the scheduling and job processing. Applications that run on grid correspond to the SaaS cloud model.

3.4 Global infrastructure

There is no global grid infrastructure, therefore environments for different use cases need to be adapted by using runtime execution environments. There were tendencies in the past for a global grid approach, but resulted normally in joining two different systems – UNIGRID (access to gLite and globus) [1], DATATAG [68], P-GRADE portal (access to Unicore and Globus) [41] or GRIP [69]. Many projects nowadays are trying to find a solution for automatically extending grid resources on demand with some external cloud resources, especially in peak times when local resources are overloaded. For now no global solution is available.

4

The Benefits and Challenges of Grid and Cloud Integration

4.1 The benefits of Grid and Cloud Integration

Grids lack the ability to adapt to exact end-user demands, such as exact version of an operating system or a precise environment for their applications. To some extent those problems are addressed by using runtime execution environments [70] or by pilot jobs [71], a use of a custom software stack on top of the existing middleware layer, such as PanDA [72], glideinWMS [73] , DIRAC [74], SoftENV [75]. Extending grid resources with cloud resources or grid virtualization is attractive from multiple aspects:

- Portability of applications – there is no need to test applications and programs in different environments, since cloud can provide an exact version of the operating system and software, which is required for a certain workload.
- Efficient resource utilization, no over-provisioning, since in grid resources are always available and may idle and in cloud resources are available on demand and leased only until needed [76].
- Scalability – cloud scales up and down easily and it scales on demand.
- Unified operating platform for users and applications – virtual machine images can be adjusted for a certain discipline or virtual organization. User can define software and execution environment.
- Higher overall availability and reliability.
- Improved system security and availability – virtual machines and applications are running separately and do not influence each other's performance.
- No maintenance and operational costs due to leasing remote resources, they are covered by the resource provider.
- More efficient operation of hardware, since multiple virtual machines can be launched with different software [20].
- Flexibility allocating hardware.
- Access to raw computing resources.
- Full host separation from the virtual machine operating system.

- Reliability - higher reliability compared to grid, while cloud follows a business model and has to follow SLA agreements, grid follows the best effort support.

4.2 Main Grid and Cloud Integration Difficulties

Computational needs are present in every discipline, but researchers lack the grid education and tools [62]. Grid infrastructure with its specific authentication and authorization procedures, job submission and execution is quite complex to use. Users are limited with predefined execution environments and are consequently obliged to test new applications in many different environments. High heterogeneity of hardware and software means that availability of the infrastructure is limited. For applications to be portable to multiple infrastructures, long development is in place and high costs involved. Cloud in this segment can solve many of those problems. With its business model it garantees high availability, EC2 99.95% in 365 days with SLA and leased resources. Opposed to expensive supercomputers, clouds promise an economical usage of leased resources and with SLA-s higher reliability than grids [77]. One of the main advantages in using cloud, is that with virtualized nodes, a user can modify her execution environment and can access newly added resources via API or a web interface. On the other hand, when running scientific applications in the public cloud, several problems can emerge. To name a few, public clouds have proprietary nonstandardized API-s, different image formats, simplified authentication mechanisms, different network connection possibilities, different contextualization of the virtual instances, accounting, not to mention problems with data management and its integration with cloud storage. Big data is a challenge for cloud architecture [78]. Where should the data be stored, how can it be accessed, should it be transferred to the cloud? Storage is usually separated from cloud and data transfers in public clouds are expensive.

4.2.1 Authentication and Authorization

As for user management, grid uses a virtual organization (VO) based system. A user authenticates using her x509 certificate, granted by a national certificate authority. The VO defines security policy; each user is a member of one or more VO-s. For access to grid, a user generates her proxy and submits jobs to multiple clusters. SSL is used for transfers. In the cloud, authentication is based on a single-sign-on username and password or on x509 key pairs and access to the cloud is granted via web portal. Since the user management in grid is used across multiple domains and achieves a certain level of security, it

should also be used in the integrated models. For private clouds, two solutions are available that solve the authentication problem: Perun for OpenNebula [79] and VOMS plugin for Keystone in OpenStack [80]. They both enable VOMS-authentication in the private cloud. For grid and cloud integration software standardization is crucial, so is formalization of application environment, registration of cloud resources in the information system and replacement of runtime environments with virtual machine images.

4.2.2 *Standardization of Software*

Within a grid community a considerable effort is put into standardization of software – organizations like Open Grid Forum (OGF), Organization for the Advancement of Structured Information Standards (OASIS), World Wide Web Consortium (W3C), Internet Engineering Task Force (IETF), Siena [81] [82] want to facilitate combining different architectures and solutions. Standardization of grid covers job execution, security, data operations and information system. At the moment there are two approaches to standardization: gateway approach, such as interoperability of gLite middleware and GOS [83]. One can submit a job or access data by both middleware solutions. A job, written in JDL (Job Description language), used in gLite, translates into JDSL format (Job Submission Description Language), used in GOS. Another approach is real standardization, where using same open standards attains interoperability. One such example is standardization of job execution using gLite, ARC and Unicore, all using OGSA-BES standard [84]. With the same user interface, job can be submitted to different grid middleware. Projects for cloud interoperability are rare - OCCI-WG, RESERVOIR, NUBA [41] [3] [85]. Cloud users will sooner or later demand common protocols for ease of federated usage. Open Grid Forum (OGF) formed an OCCI working group that focuses on development of common API-s for cloud services, such as OCCI [82]. Distributed Management Task Force supports development of standards for distributed infrastructures, such as Open Virtualization Format (OVF) or Cloud Infrastructure Management Interface (CIMI). For cloud interoperability OpenID, Odata, CDMI, AMQP and XMPP standards can be used [86].

4.2.3 *Data Management*

Clouds were developed for hosting services on demand, not for computing tasks. Network and disk storage is not properly addressed in the cloud. Storage capacities are usually entities outside of the public cloud, consumer has to pay for data transfer from the

storage to compute node. Network connections are slow and access mode and security to the public cloud different from cloud to cloud. Some enable enhanced network features (such as clustering), but come with a higher price to pay. Performance and price of data-intensive tasks in the cloud is one of the biggest challenges.

4.2.4 Cloud Instances Provisioning

Cloud provisioning is a multi-layered problem. The provisioning entails starting and stopping cloud instances on demand. How to detect high demand and launch instances automatically? When to stop the machines? How to limit resources and which jobs should be transferred to the cloud. Then the creation of cloud instances has to include a definition of a security policy, preparing firewall rules, choosing a suitable VM image and instance type. When integrating grid and cloud, compromises have to be made on which architectural procedures will be adopted, what kind of authorization and authentication techniques will be used, where will the data be stored and how privacy ensured, what will be the job submission mechanism, what kind of runtime environment deployment etc.

4.2.5 Performance penalty

Multiple virtual instances are hosted on the same hardware and operating system. In the IaaS cloud, each VM is running its own OS and is isolated from other VM-s on the same host, but they all share physical resources with other guests. If for example a full backup is running on one instance, it will deteriorate the overall system performance, so consequently the performance of other virtual machines. The solution is in per-VM limits, but in the cloud, resources are usually over-commited, meaning that if one VM does not use all the requested resources, another instance, that has higher resources need, will use the free resources from the first one. Noisy neighbors, applications or VM-s that consume a lot of resources and produce a high load, have an effect on other VM performance. CPU and memory caching is not as efficient as on a bare metal server, while multiple tenants are accessing the same physical resources and cause CPU interrupts, I/O wait, network I/O etc. Even though cloud performance is lower than the performance of a physical node, it can still benefit to use cases where resources are needed immediately and temporarily.

5

Interoperability and Integration

Interoperability indicates that two systems can work together, despite the fact that they were developed independently. Interoperability of grid and cloud systems therefore signifies that grid can harness cloud resources via a unified interface. When discussing the interoperability of grid middleware solutions, the aim is to enable two different systems to cooperate and exploit each others' functions and services. This interaction between two or more systems is based on open standards [1]. Two different and independent technologies can only work together on the basis of standards. Standards define protocols, specifications, usage predictions etc. If two systems do not follow the same standards and are not interoperable (which is the case with grid and cloud), their integration is needed.

Definition 5: Two or more systems are interoperable when they can work together, even if they were developed independently, due to their standardization. The systems can make use of each others' functionalities.

Since grid solves some major academic and research problems, some of the grid techniques should be encompassed in the cloud as well. Those techniques cover data transfer, resources and job monitoring, job scheduling and user interface, user authorization, VO management and security mechanisms.

Definition 6: Integration is a procedure of joining two or more non-interoperable systems physically or functionally into one new system.

Integration of grid and cloud demands a precise analysis of both technologies, identification of disadvantages, difficulties and common points of usage, migration and integration. The first question in place is whether cloud should be complementary to grid or as its substitute. The main challenge of cloud is big data [44] (more than 1 PetaByte). Where should the data be stored, how should it be transferred, should it migrate from grid to cloud? Is cloud designed for high data transfers and big data storage? Network transfer and disk space is expensive in the cloud. Should cloud only be used in peak times? Because of the data challenge of the cloud it would be best to start running applications with low input/output needs in the cloud. The focus should be on minimizing the performance loss. Another big issue is the implementation of grid authentication mechanisms in the cloud. The current cloud authentication is based on username, password and credit card number. It is simple, user-friendly, but not very secure. How would

the access to the cloud be controlled, limited, authorized and how will the VO system be implemented? Grid middleware is not aware of any cloud resources at the moment - cloud resources should be discoverable via common information system so that clients can interact with them and that grid will be capable of hosting cloud applications and instances. Schwiegelshohn et al. [1] distinguish between four types of interoperability:

- User interoperability which enables user migration between different virtual runtime environments.
- Resources interoperability which enables sharing of same resources between different VO-s.
- Design interoperability, determining a common interface for accessing all grid systems and which includes service, authentication and security interoperability, pointing out the transfer of design concepts between different runtime environments. In this case the user can be authenticated, is authorized and transfers his data in the infrastructure the same way, even if different middleware solutions are applied on the system.
- Data interoperability, necessary especially in interdisciplinary studies that enables access to different distributed databases for same users. This type of interoperability requires common data and metadata formats.

Integration indicates that independent components or systems are joined into a bigger system that functions as a coordinated entity. One system can read or use the data managed by another system. Integration of grid and cloud would therefore represent joining both technologies into a new architecture. Ordinary problems when switching from grid to cloud include lower performance due to virtualization of hardware (another layer), network limitations and drawbacks, unanswered demands for high volume of communication of MPI jobs, slow and expensive data transfers and the fact that also cloud is not unlimited [3].

Schwiegelshohn [1] suggests the migration of grid services from middleware to the operating system layer - accounting, authorization, monitoring and security would become independent services. Then a common operating system should be used for all VO-s, like XtreemOS [87] [88] to abide resource management. XtreemOS automatically configures user credentials, chooses suitable resources and starts an application.

Main grid and cloud interoperability problems can be organized into 5 groups:

- resource brokering – scheduling,
- standardization,
- runtime environments or image catalogues,
- data storage and transport,
- access to resources and security.

5.1 Scheduling with information system

Information services and scheduler walk hand in hand. Information services provide information about the state of resources, their availability, types etc. Since not all jobs are suitable for all machines, a scheduler chooses the appropriate resources, based on the job description. Managing resource and job scheduling in cloud is challenging. How can resources be dynamically allocated and registered? How can a high demand be detected? How can the quality of service be ensured? Creating cloud resources in advance, so that they are acknowledged by the resource broker leads to over-provisioning and is killing the basic idea of the cloud infrastructure usage. Cloud resources are therefore available in advance, not by demand. Current schedulers focus on CPU usage and fair shares, ignoring the I/O, which is a critical point when scheduling virtual machines. Virtual machines have their own operating system and software, therefore internal resource management. Disregarding I/O usage and network behavior of a virtual machine can cause network congestions and high I/O wait which can lower the overall performance and throughput on the physical machine [89] [90]. Scheduling is very important, tasks can be transferred to nodes that have lower load and can be automatically resubmitted if they fail.

5.2 Standardization

We discussed standardization problems briefly already in the Chapter 4. A lot of effort is put in the standardization of the software used in grid, less interest is shown for cloud standardization. But customers will demand standardized API-s and protocols to harness federated usage of multiple public clouds providers. OGF has an OCCI group, developing common API-s for cloud services [41]. Open Grid Forum standards include JSDL (Job Submission Description Language), which is used in gLite middleware and was implemented into ARC client within EGI InSpire project between 2010 and 2014

[91] and European Middleware Initiative (EMI) [26]. Also BES (Basic Execution Service) standard was accepted and is implemented into ARC middleware. Data standards are OGSA (covering data transfer, data access, storage, cache, replication and management) and GRIDFTP transfer protocol, both implemented in ARC.

5.3 Runtime environments and image catalogs

Grid infrastructure lacks a common flexible environment which are provided in the IaaS cloud. Running applications in cloud environment demands customization of the job description and application itself, which is conditioned by possessing special technical skills. VRE could be integrated into virtual machine images, therefore RTE-s could be omitted. There is a downside in cloud image catalogues, namely that each private or public cloud provider has a different set of images, those images are not interoperable [64].

5.4 Data storage and transport

Data is distributed in grid environments. It is usually distributed in multiple copies, so the nearest data store is chosen, when running jobs on a grid. Because of the distribution, data transfers are frequent. In grid, a reliable, secure and fast protocol is used for data transfers, ftp with grid security interface (GRIDFTP). Optimizations in data access are possible. ARC middleware, for example, uses data staging and data cashing to improve performance, ATLAS Experiment [19] uses CVMFS [92] as a software repository.

5.5 Security and access to the grid

Security is a critical topic in cloud computing. At the expense of ease of use, a certain level of security is lost. In cloud, VM provisioning is done with a username and password, access to the instance via ssh is realized by an ssh-key pair. Credentials are different in every public and private cloud. For improving security without reducing practicality experts suggest a single-sign-on (SSO) for the user management [41]. Grid on the other hand has authorization based on virtual organizations (VO). This system can be used across multiple domains and defines security policy. User obtains a certificate, granted by a CA Authority, generates a user proxy based on his private key pair, which is then used by grid security infrastructure (GSI) and for secure transfer. Privileges can also be delegated to a middle service, called workload management system (WMS) [24], which could be

seen as a security hole. On the other hand the creation of proxy is advantageous in comparison to SSO, while it is not necessary to rewrite private key password each time, nor it is necessary to cache it. A proxy is usually granted on a machine that has a user private key. The more complex approach to security in a hybrid grid and cloud infrastructure is adopting the VO-based authorization. Security is not the only reason. VO-s enable collaboration among multiple organizations or clusters and enable resource sharing. Each user can be a member of multiple VO-s. VO-s are supported on different clusters, they provide usage policy and share resources between members. Those resources can be the data, software installed, special hardware etc. The advantage of VO-system is also the feature to hide the complexity of grid to its users [24].

6

Grid and Cloud Integration Possibilities and Available Solutions

Approaches to grid and cloud integration vary, depending on the interpretation of grid and cloud, software used, access rights, executed tasks etc. Some solutions are already available, but they usually correspond to a specific use case or can only be used for a specific public cloud provider.

Two different and independent technologies can only work together on the basis of standards. Standards define protocols, specifications, usage predictions. If two systems do not follow the same standards and are not interoperable (which is the case with grid and cloud), their integration is needed. Since grid solved some major academic and research problems, its techniques should be encompassed in the cloud as well. Those techniques cover data transfer, resources and job monitoring, job scheduling and user interface, user authorization, VO management and security mechanisms. Integration of grid and cloud demands a precise analysis of both technologies, identification of disadvantages, difficulties and common points of usage, migration and integration. The first question in place is whether cloud should be complementary to grid or as its substitute. The main challenge of cloud is big data (more than 1 PetaByte) [78]. Where should the data be stored, how should it be transferred, should it migrate from grid to cloud? Is cloud designed for high data transfers and big data storage? Network transfer and disk space is expensive in the cloud. Should cloud be used only in peak times? Because of the data challenge of the cloud it would be best to start running applications with low input/output needs in the cloud. The focus should be on minimizing the performance loss.

Another big issue is the implementation of security mechanisms in the cloud. In the public cloud, a username and password are provided for VM provisioning, access to the instances is enabled by an SSH-key. It is simple, user-friendly, but entails a low level of security. Another problem is defining a security group for the instances. By default only SSH port is opened from external networks. To avoid accessibility problems, users enable open access to their instances, which is an important security flaw. How would the access to the cloud be controlled, limited, authorized and how will the VO system be implemented? Grid middleware is not aware of any cloud resources at the moment - cloud resources should be discoverable via common information system so that clients can interact with them and that grid will be capable of hosting cloud applications and instances.

	CLOUD	GRID
SaaS	gmail, mysql	NAMD, ESPRESSO
PaaS	Google engine, AWS	OPENMPI, LRMS
IaaS	OpenStack, EC2	monitoring, accounting

Figure 6.1

Grid and Cloud comparison: both architectures enable access to bare metal servers. Grid services belong to the PaaS layer since they provide interfaces to the resources, applications running on grid correspond to the SaaS layer.

6.1 Models of Integration

Different models of integrating grid and cloud exist, depending on the abstraction of both technologies, on the cloud definition and on the use case. Grid services are actually in the PaaS layer since they provide interfaces to use resources (see Fig. 6.1). In a PaaS cloud scheduling is transparent to the user and is done in the background, in grid user has control over the scheduling and job processing.

In order to find an appropriate integration model, we need to answer the following questions: does integration propose deploying a cluster-on-demand or extending an existing grid cluster with virtual resources that are automatically provisioned? Where are those additional resources available - in the public cloud, in the private cloud or in another distributed infrastructure? Does integration include common access to two or more different infrastructures by a unified user interface? What kind of user action is necessary for using the integrated infrastructure, what are the procedures, what is the final goal? Cluster-on-demand is normally based on a public or private cloud and is not integrated with grid infrastructure. It provides additional virtual resources when needed. Virtual worker nodes are managed by a batch system. These solutions focus on web portals, from which a user can easily monitor and manage virtual resources. Some open source solutions are available, such as Elasticluster [93], STAR cluster [94], Science Cloud [95] or SCMS.pro [96]. The concept of "grid in the cloud", where an entire cluster and all grid services are pre-configured in VM images, enables clients to communicate with grid services via grid protocols. When additional resources are needed, a grid cluster is launched on the public or private cloud.

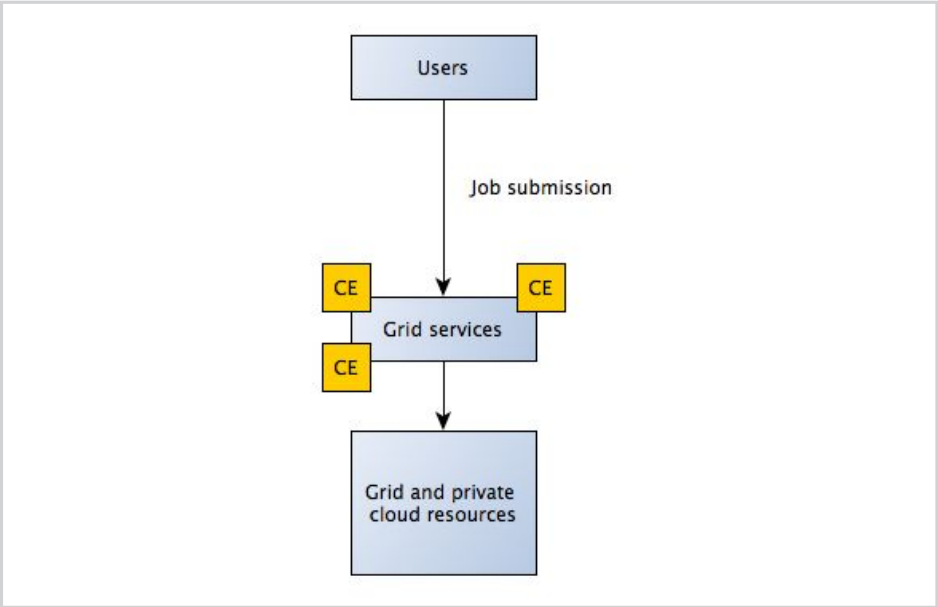


Figure 6.2

Integration of grid and private cloud.

6.1.1 Grid and Private Cloud Integration

This approach of grid and cloud integration is sharing data and physical resources across grid and private cloud infrastructure (see Fig. 6.2). To execute the workload on a certain infrastructure, a user has to adapt the code and execution scripts and choose appropriate infrastructure for her tasks. When cloud is chosen, a local copy of the data has to be created on the virtual machine. One virtual machine is used per task, it is terminated when the task is finished. Compute elements need to be modified in order to be capable of managing virtual machines and grid cluster via grid manager. To be recognized by the grid manager, worker nodes and storage need to be virtualized.

This model has been implemented in Italy - WNoDeS framework (Worker Nodes on Demand Service) [97] where worker nodes of a Linux cluster are virtualized on demand. Computing and data resources are virtualized and adjusted. Grid manager (LRMS) [98] controls the virtual machines. This is not grid in the cloud, or cloud in the grid. It is a bare virtualization of grid resources. The main service is a bait server, a virtual machine that works as an LRMS. Virtual worker nodes are added on demand. Three interfaces are available for job submission: local, grid and OCCI. The main advantage of this solution is the possibility of using virtual machines for job execution, however there is an

important constraint - neither a VM-specific job can be executed on a physical worker node, nor a conventional job can be executed on a virtualized node. Adding to that, the whole process is not automated, administrator intervention is required for each VM execution. Another similar solution is ARC RAINBOW [99], where choosing an appropriate runtime environment launches a virtual machine with preinstalled software, that enables job execution under a Windows operating system. The solution is used by Ukranian Medgrid VO. Job is monitored by accessing the virtual machine via remote desktop protocol. The described solutions solve the problem of customized runtime environments in a simple way but do not help when conventional resources in the cluster are fully occupied and additional resources are needed. Another drawback is that the procedure is not automated, resources need to be manually assigned or requested and user execution scripts need to be adjusted. The same could be solved without virtualization, using chrooted operating systems (CHOS) [100].

6.1.2 Unified User Interface to Access Grid and Cloud infrastructure

The second approach is an alternative to the first one. Cloud and grid infrastructure remain unchanged, modifications are made to the user interface (see Fig. 6.3) The user interface overtakes the tasks of compute element, such as managing execution, monitoring jobs, controlling input and output data etc. User runs her jobs in a virtual machine. Execution environments are omitted or used to describe the image to be used for VM creation. Job description includes software stack, which is deployed when a bare virtual machine is created. Major efforts in grid community resulted in appearance of different interoperable grid clients and interfaces. The same client can be used for submission to different grid clusters, supporting different grid middleware. To mention a few: GridX1 [8], GridWay [37], EMI-ES [26], DIRAC [101]. These solutions enable a single point of entry to grid (or several different grids) and cloud infrastructures. However different providers define instances differently. Which instance will be launched and how will it be registered in the information system? Predefined instances in information system can cause additional mess. Data transfer is a problem that needs to be addressed too. Since the whole software stack is sent with the job, huge data transfers are required. Data transfer in the public cloud is expensive. There are a few interoperable clients available, that enable job submission to grid and cloud infrastructure. Swarm [102] [103] is a web interface, used to submit jobs to Amazon EC2 and TeraGrid. Cloud resources are used for shorter and non- demanding jobs, grid for all the other. A user interven-

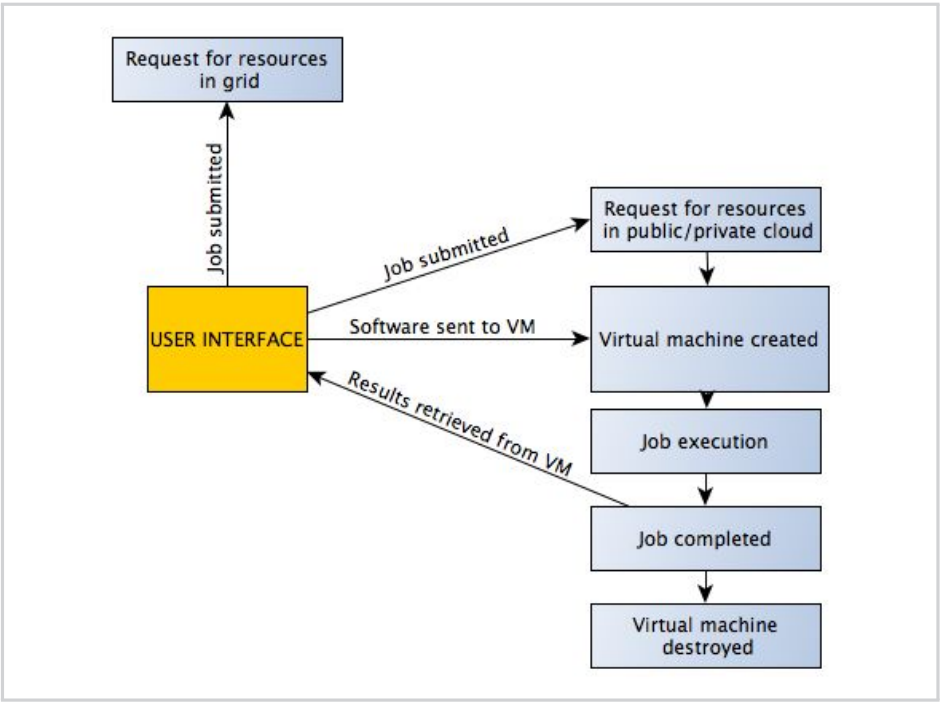


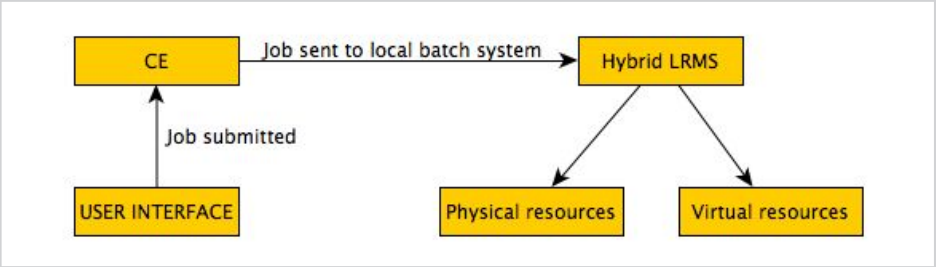
Figure 6.3

Grid and Cloud accessed by a unified user interface.

tion is required when choosing the suitable platform for job execution, it is not selected automatically. In long term the ability to monitor and control a large number of jobs manually is not feasible. XtremWeb [104] tries to join both architectures in the context of volunteer computing. Users can write their own applications and use them in grid or cloud. When submitting a task, a user has to run submission scripts manually and needs to decide which resource to use. Furthermore jobs can be executed in Amazon EC2 or Venus-C cloud [105]. The major drawback of this solution is that job submission is not automated. Cloud and grid resources are manually chosen, user scripts manually adjusted, therefore user intervention is always required, not to mention that grid resources are highly limited while available on volunteer basis.

6.1.3 Grid and Public or Hybrid Cloud integration

The third integration approach is grid and public cloud integration (see Fig. 6.4), where user's execution scripts and workload are preserved and no modifications of the code are necessary, grid is only expanded to the cloud when additional resources are required



6.4
ı of grid and
ıd.

(cloud bursting). To achieve this kind of integration, local batch system (LRMS) needs to manage cloud and grid resources. This model is complex to implement. It enables usage of grid and cloud resources, accounting, resource monitoring on both architectures, elasticity, both cloud and grid storage usage, however it requires virtualized worker nodes, adaptation of LRMS, customized execution environments, implementation of compatible authorization and authentication mechanisms on both architectures, implementation of accounting and billing system and virtualization of some central grid services [106]. In this model user applications/jobs are meant to run as virtual machines. This solution enables integration of grid and cloud, resource sharing between grid and private cloud, resource usage of grid and hybrid cloud, optimal resources consumption and a solution for the peak times, when public cloud resources can be available via unified interoperable interface.

At the University of Goettingen a hybrid cloud with Unicore grid middleware has been deployed [44]. Central grid services run in the public Eucalyptus cloud, which is extended, to the AWS public cloud. Another interesting solution is Science Clouds project, making use of Nimbus [95], that can provide a virtual cluster for grid applications, such as a batch system or a workload manager. In this setup a client requests a resource lease for a few hours and if the request is authorized, a virtual machine is deployed. A client is then allowed to configure the VM according to its exact specification and is given an exclusive ownership of the leased resource. This project implements a number of interesting features, such as private IP addresses for virtual machines and network virtualization, which enables the possibility of deploying cross-domain virtual clusters. At CERN [107] an OpenStack private cloud was deployed with HTCondor and Cloudscheduler [108] and computations are executed on cloud instances. Their goal was to use grid and cloud as complementary services. Cloudscheduler is used to manage virtual machines, HTCondor to manage batch jobs. Cloudscheduler talks to HTCondor

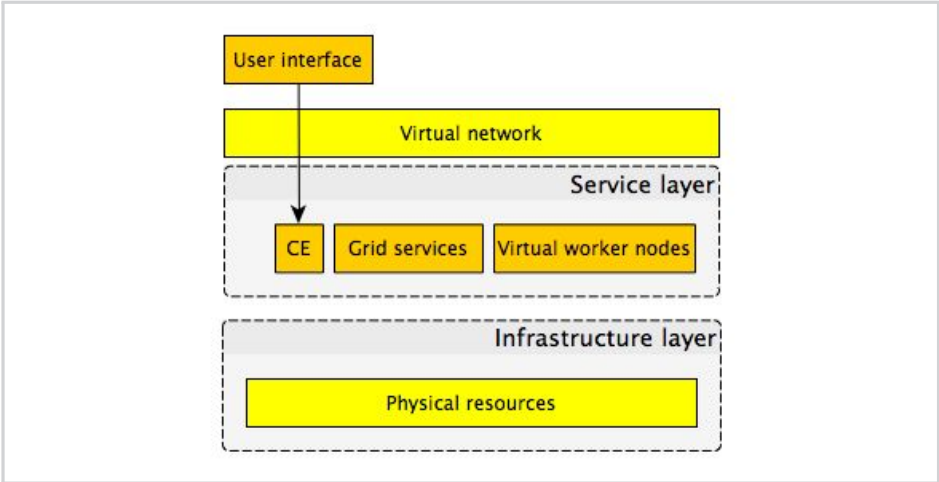
about requested resources for jobs in queue and creates virtual machines. Resources are dynamic, they appear and disappear on demand. After the job is finished, the virtual machine is destroyed. The solution solves the problem of using certain versions of operating systems, software and libraries for old dataset, although their developers ran into certain problems. There is no mechanism for scheduling and monitoring jobs and job states in the cloud, the system is unable to reuse a cloud instance after the job is finished (which is an overkill), CVMFS [92] and proxy discovery, performance is deteriorated due to virtualization and a lot of resources are wasted. In fact this is not grid and cloud interoperability. It is grid workload, adapted and run in the cloud. Amazon integrated cloud and HPC providing a so-called HPC cloud, in combination with EBS data storage [109]. This solution allows multiple accesses to the data storage, storage is available longer than instances themselves.

6.1.4 Federated clouds

Some efforts were made to join clouds into a grid (Grid-of-clouds), federated clouds, such as Nimbus [110], StratusLab [111] [112], Aneka [113], CloudSigma [114], Helix Nebula [115] or to deploy a virtualized grid cluster (see Fig. 6.5). In this case grid authentication procedures are preserved, access to the cloud resources is simplified, enabled directly with a cloud API. User workload has to be adjusted, APIs and submission methods as well. Venus-C developed and deployed a PaaS cloud for research and industry communities. The goal is to enable porting of grid workloads into cloud, providing the necessary software and applications. Those infrastructures are used for CPU-intensive jobs with low data and network requirements (simulations). All grid services and worker nodes are virtualized. It is a federation of grid clusters in the cloud.

6.2 Possible Solutions to Grid and Cloud integration problem

After studying the available solutions, software, middleware and interware and defining our objectives, we have tested and evaluated 6 possible integration models: (1) cluster of virtual machines with Elasticluster [93], (2) cluster of virtual machines with STARcluster [94], (3) Cloudscheduler [108] with HTCondor, ARC and OpenStack, (4) ARC and OpenStack/EC2 using SLURM Cloud Plugin, (5) ARC and OpenStack with SLURM prolog scripts and finally (6) Virtual ARC cluster in the Cloud. Each solution has its advantages, we present the models in the next sections. For the final integration model



6.5

ı of grid and
ıd by virtualizing
ıster

we have developped the last solution, ARC cluster in the Cloud, which is described in detail in Chapter 8.

6.2.1 *Elasticluster*

Elasticluster [116] [93] is a tool that enables creating a cluster of virtual machines in the cloud. By customizing a text file, nodes can be added or removed from the cluster, virtual machines created and software can be installed and configured. Private and public cloud can be used. Elasticluster connects to the cloud (private or public – OpenStack, Amazon EC2 or Google Grid Engine), starts virtual instances and waits until they are accessible by ssh. Virtual machines are configured using Ansible [117], software needed for the task execution is installed. One of the virtual machines is set as a master and runs a batch system that controls other virtual machines in the cluster (see Fig. 6.6). Supported batch systems are SLURM, SGE and Torque. When cluster is created nodes/virtual machines can be added automatically. Implementing this solution to our integration model seems unsuitable while the process of creating the whole virtual cluster from the spot seems time consuming expert work, similar to setting up a real cluster. When cloud resources are available permanently this is feasible, while the virtual machines configuration and software installation is a one-time process. The recreation of the cluster in each peak time, does not seem as a convenient solution. This solution solves the problem of creating custom execution environments and enables batch job execution in the cloud in peak times, when there are higher demands for resources. The procedure is not automated,

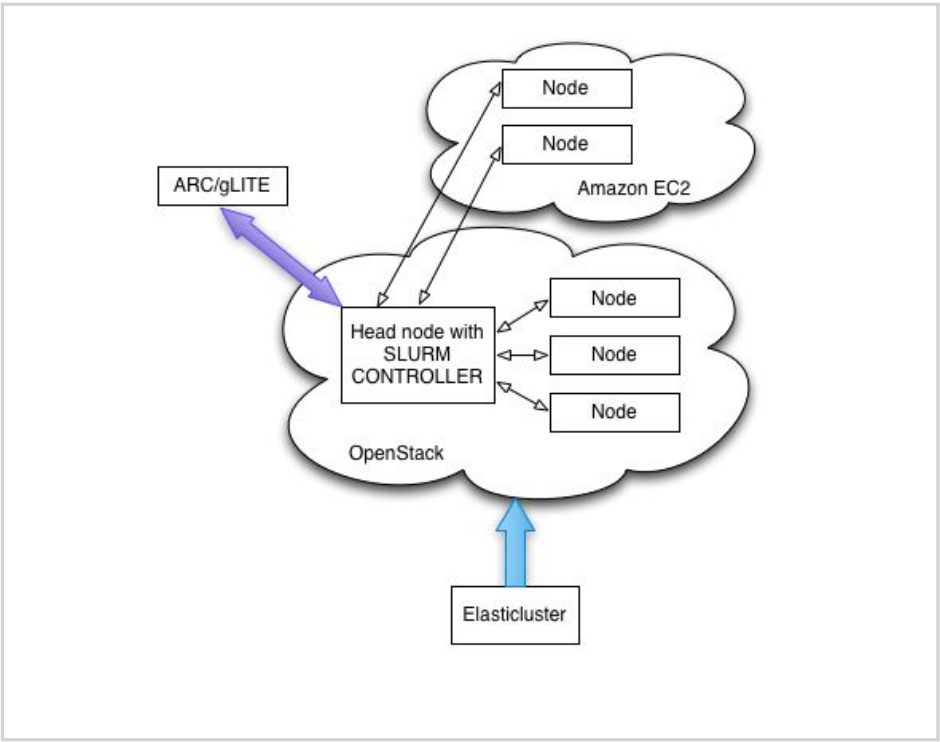


Figure 6.6

Grid and Cloud i
by using Elasticlu
cluster in the clou

whereas the configuration text file needs to be modified to add or remove virtual worker nodes from the cluster. To be used in our existing infrastructure, all grid machines need to be virtualized, being a part of an OpenStack cloud and managed via SLURM. Grid middleware should be incorporated in the integration model. Some similar solutions are available – StarCluster [94] creates a virtual cluster in the cloud, but only supports EC2 Cloud. Bright [118] supports most major batch systems and clouds, but is a commercial solution. It is rather using virtual machines for scientific workloads, not grid and cloud integration.

6.2.2 STARcluster

The second model is to run ARC with Open Grid Scheduler [47], Amazon EC2 cloud and StarCluster. StarCluster (Software tools for Academics and Researchers) is a toolkit for launching and orchestrating clusters of virtual machines on EC2 via boto API [119]. It is a similar solution to Elasticluster, but it only supports Amazon EC2 cloud. Virtual

machines are managed by a local batch system. It is oriented more toward an SaaS cloud, since it enables deployment of software stack as a virtual machine. The problem with this solution is that OpenStack is not supported, only preconfigured images are available and it can only be used on Amazon EC2. Our scope are different public and private clouds.

6.2.3 *Cloudscheduler with HTCondor and OpenStack*

The third possible solution is the usage of Cloudscheduler with ARC, HTCondor and Openstack/EC2. It is enabling grid workload to be executed on different public and private clouds. A user submits a batch job to HTCondor, which communicates with Cloudscheduler about the required resources for the virtual machine. The Cloudscheduler manages virtual machines. Resources are dynamic, they appear and disappear on demand. After the job is finished, the virtual machine is destroyed. There is no mechanism for scheduling and monitoring jobs and their states in the cloud. A virtual machine is launched for each job, the system is unable to reuse a cloud instance after the job is finished which is an overkill.

6.2.4 *SLURM Elastic Cluster with SLURM Cloud Plugin*

This is a cloud bursting solution since it enables running of batch jobs in the cloud. SLURM demands all nodes to share SLURM configuration, have specific description of each node and its resources in the configuration file and have a defined public IP address and hostname. Multiple public cloud users cannot be used for virtual instances creation. SLURM supports cloud features. A separate partition is created in the batch system and is tagged by the Cloud feature. Virtual resources are also managed by SLURM. The inconvenience is that virtual machines have to be up and running, no modifications can be made on the machine by user. VM need to be assigned with the same IP and hostname and defined in the configuration and hosts file. No resources can be added automatically, administrator has to change the SLURM configuration file, every time new machines are added to the cluster.

6.2.5 *SLURM Extension to the Cloud using Prolog Scripts*

The fourth model includes virtualization of worker nodes with OpenStack in combination with ARC and SLURM prolog scripts (see Fig. 6.7). This solutions demands the least of change on the grid cluster. A batch job launches a virtual machine and runs there. Compute elements need to be modified in order to be capable of managing virtual

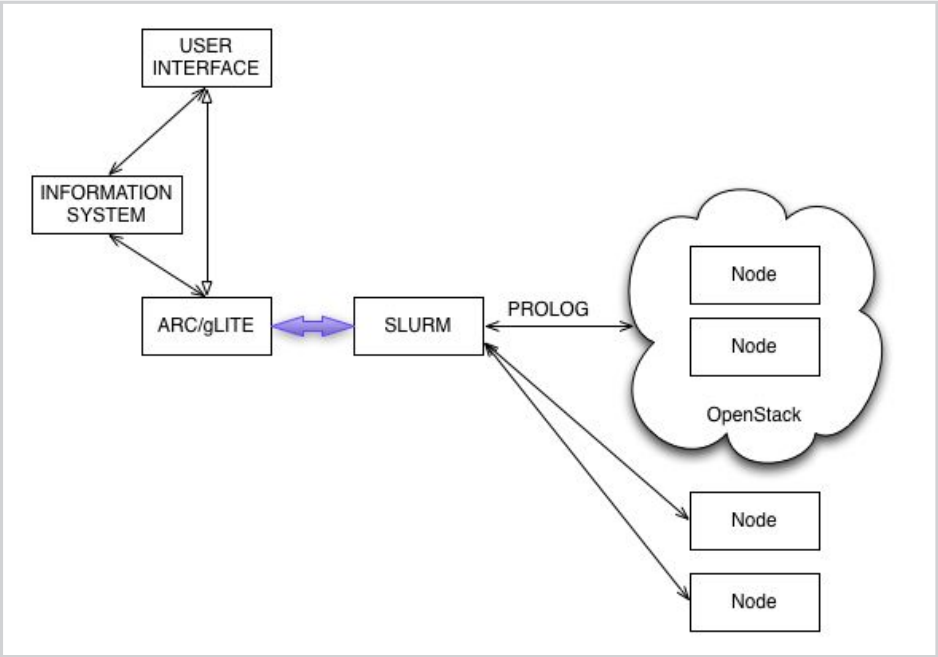


Figure 6.7

Running batch jc
cloud by SLURM
scripts execution

machines and grid cluster via the grid manager. To be recognized by the grid manager, worker nodes and storage need to be virtualized. When extending private cloud to the public Amazon EC2 cloud, credentials are needed to create cloud instances. In order to support this option, Arnes [120] would be obliged to implement a billing system. SLURM Prolog scripts could be adapted to support multiple EC2 usernames, but Prolog scripts cannot contain users' passwords. Users cannot modify VM configurations; all worker nodes need to be virtualized to support the VM creation. One VM is used per task. This model does not solve problems of small VO-s or individual researchers, since it does not offer the possibility to run their workload on different public clouds. Slurm prolog scrips could be replaced by chroot operating systems (CHOS) or chroot environments [100].

6.2.6 ARC cluster in the Cloud

This solution requires full virtualization of the grid cluster and run it in the cloud. Setting up the whole cluster in the cloud is time consuming, but solves the conventional

constraints in the grid, while user execution environment can be fully deployed (runtime environments are replaced by virtual machine images) and the cluster can scale easily. In the private cloud, where data store can be part of the virtual cluster, data handling is covered, for public clouds, some other approach should be implemented, while transferring several GiB of data from and to the data store and from and to the cloud does not seem to be possible in production environments.

Approaches to grid and cloud integration vary - a deployment of a virtual grid cluster in the cloud is one option, VM usage instead of RTEs, cloud resources as an extension of the conventional grid cluster to the cloud (cloud bursting) or unified interfaces. From the enumerated solutions, we have developped the last one, deploying an ARC Cluster in the Cloud - ARC-CC). This solution eases the implementation of several functionalities that are indispensable for our users and projects, such as monitoring, accounting, registration of the services and jobs in the information system etc. It is compatible with all public and private clouds. Using adaptive execution environments, Prolog scripts or Chrooted environments does have an advantage, because they enable a customization of execution environments and an extension to the cloud, but they do not include other functionalities, such as accounting or job status monitoring. In our implementation we would like to achieve the following:

- automatic creation of virtual machines in peak times,
- VOMS-based authorization,
- usage transparent to user,
- preservation of the established procedures for job submisson and no modifications of users' workload.
- resources in the cloud are not reserved or permanently allocated, but used on demand.

We propose integration of grid and IaaS cloud. The IaaS layer enables resource management interface to deploy virtual machines with specified operating system and software installed. Both grid and IaaS cloud give access to bare metal resources, grid has already preconfigured software installed, cloud has an interface to deploy operating system and applications on it [121]. Virtualization is problematic from the performance point of

view, but attractive at the same time while it separates virtual machines from the host machine and can adapt to exact end-user demands. This approach can provision worker nodes on demand and can provide a reliable and stable hybrid infrastructure. Cloud resources should be available with minimal management effort.

Our primary goal was to integrate grid with hybrid clouds via an interoperable LRMS, that would be able to manage virtual and physical resources. This solution is suitable for an integration of grid and private cloud, otherwise, it could only be used for a single public cloud provider. Cloud providers use their own software and API-s that cannot be used for other cloud solutions. Open source solutions that address those problems are rare and do not solve federated usage at the moment. By grid cluster virtualization, we are not limited to a single cloud provider, but can build a virtual cluster on any private or public cloud.

Current Grid and Cloud infrastructure

7

The Slovenian NREN, Arnes [120], provides HPC and grid clusters to researchers in academia and science. We currently support two separate infrastructures. The first is a hybrid grid that combines gLite and ARC grid middlewares (majority of jobs are submitted via ARC) and the second hybrid cloud infrastructures that contains the OpenStack [60] private cloud with the extension possibility to the Amazon EC2. Among different open source cloud computing softwares, such as OpenNebula [58], Synnefo [122], Cloudstack [123], we have chosen OpenStack, since it is highly developed, widely used and fully supported in CentOS [124], our distribution of a choice and supports a lot of features that are not supported in other cloud software solutions, such as object storage or complex network configuration with vendor support. Grid users are all members of various virtual organizations, while Arnes is a member of the European Grid Initiative [91], our users are also eligible to use resources of other clusters in the EGI community, depending on where their VO-s are supported. The infrastructure at Arnes was used for testing different integration possibilities and for the testbed of the developed solution, described in this work.

7.1 Current Grid Infrastructure

Our grid infrastructure consists of 4200 CPU cores, 10TB or RAM, almost 0.5PB of data-caching storage and 100TB of permanent data storage for users' workload. More than 3000 CPU cores are connected by a fast low-latency QDR infiniband network (40Gbps) and form an HPC cluster. We support gLite and ARC middleware. For a local batch system we use SLURM. Users' execution environments are set by using runtime environments (RTE), which are accessible from all worker nodes. We also support GPGPU computing, with K10, K20 and K40 Nvidia Tesla Graphic Cards.

Fig. 7.1 shows the structure of a hybrid grid infrastructure supporting ARC and gLite middleware and job processing procedure. A user submits a job in an appropriate job description language: for the usage of gLite middleware EMI-ES user interface can be used and ARC client for submission to the ARC middleware . User authorization is done via different VOMS servers based on x509 user certificate. After the user is authorized, his job is submitted to an adequate compute element. Job is registered in the information system and transferred to a local batch system that sends the job to the corresponding worker node. A dedicated service on each compute element queries the status of the job and after the job is done, a user can transfer the results from the compute element. For simpler usage, shared filesystems are used between compute elements and worker nodes.

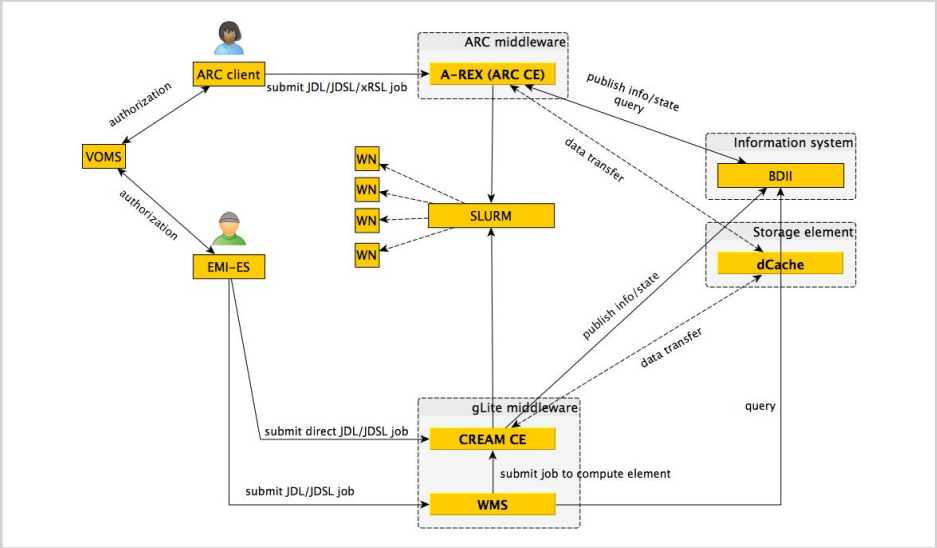


Figure 7.1

Grid infrastructure at Arnes, supporting gLite and ARC middleware.

Input files and results can be stored on storage elements, such as dCache [125], which can be used from both middleware compute elements.

7.1.1 Scheduler

ARC can be used with different schedulers, such as SLURM [126] [127], PBS [[128], SGE [129] and HTCondor [25] [130]. In grid centralized resource managers are used - following the server-client paradigm. The centralized server, usually installed on the head node, is doing the provisioning and job execution. While SLURM is used on our grid production cluster, we have chosen HTCondor for our batch system in the ARC Cloud Cluster (ARC-CC) based on our experiences and its supported features: HTCondor is free, scalable, supports accounting with APEL, is supported in gLite and ARC, does not require a database, enables job queue failover and has fully configurable partitioning, fairshare and reservations. Its advantage is that nodes are added to the cluster automatically, sharing one configuration file with the head node. HTCondor works well with HTC and HPC clusters. SLURM is less convenient for ARC-CC model, whereas it requires a unique hostname for each worker node, list of its' resources in the SLURM configuration file and has a complex user management with an underlying database. It is too complex and would cause additional problems when used in a virtualized grid.

7.1.2 Authorization and jobs submission

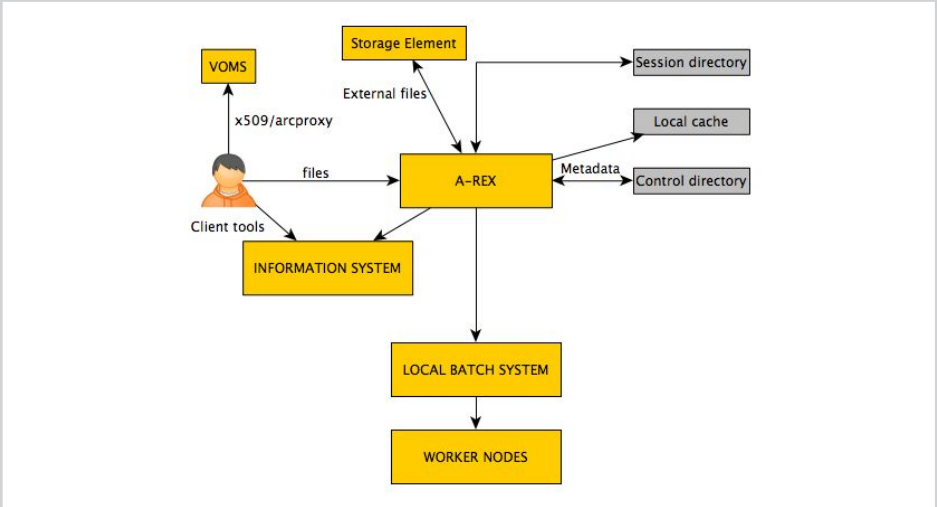
Authorization is based on certificates, issued by a local certificate authority, member of EUGridPMA [131]. A user then joins different virtual organizations (VO-s), based on the field or projects that one is working on. A VO-based approach provides a scientific collaboration management across multiple sites and it defines a joint security policy [132]. Fig. 7.2 shows the components of the ARC cluster. It consists of a Local Resource Management System (LRMS) [98], ARC compute element (ARC-CE) with a-rex, gridftpd and infosys services, storage element, VOMS and worker nodes. The procedure for job submission in ARC cluster is as follows. A user submits a job in an appropriate job description language - XRSL job description language [133] for ARC and JDL or JDSL job description language [134] for gLite. User authorization is done via different VOMS servers [7] based on x509 user certificate [135]. After the user is authorized, his job is submitted to an adequate compute element. The job is registered in the information system and transferred to a local batch system that sends the job to the corresponding worker node. A dedicated service on each compute element queries the status of the job and after the job is done, a user can transfer the results from the compute element. For simpler usage, shared filesystems are used between compute elements and worker nodes. Input files and results can be stored on external storage elements, such as dCache [125], which can be used from both, gLite and ARC middlewares compute elements. Job status can be monitored from the client side. After the job is finished, data is transferred from the worker node to the head node, the status of the job changes to 'finished' and the user can download the results from the cluster to her local machine.

7.2 Current Private Cloud Infrastructure

7.2.1 OpenStack cloud

For the private cloud OpenStack software was installed and configured. Main OpenStack components/services (see Fig. 7.3) are:

- Horizon - OpenStack Dashboard, a web-based entry point for all users. All OpenStack API-s communicate with Horizon.
- Quantum as a network service,
- Nova as compute nodes,



7.2

sion process
middleware.

- Keystone as identity service,
- Cinder as block storage,
- Glance as image service,
- Swift as an object storage.

7.2.2 *Hypervisor*

Multiple options are available for backend configuration of our private cloud, depending on the chosen cloud software stack (Xen, KVM, LXC, VMware etc.). Linux containers (LXC) use 0.5GB less RAM on Linux than on KVM. We have chosen both KVM and LXC in our private cloud. Linux containers achieve performance results, similar to bare metal servers, but they do not support to launch different operating systems (OS), such as Windows, Unix or MacOS. KVM on the other hand do have some memory and CPU overhead, but enable all sorts of operating systems and full customization of the software stack, installed on the OS.

7.2.3 *OpenStack main services*

As shown on the Fig. 7.3, three types of storages are used. Block storage provides volumes for compute nodes and is most suitable for input/output data and caching, since

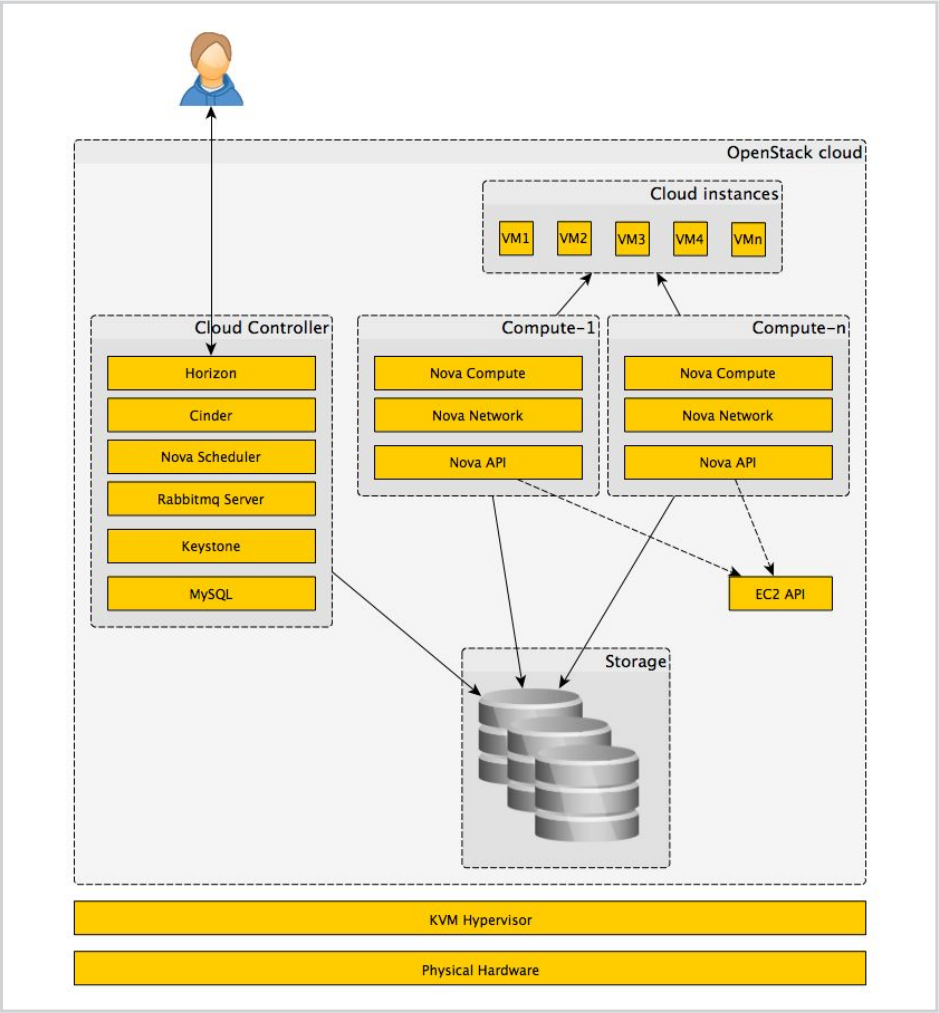


Figure 7.3

Private cloud usir
Stack at Arnes.

application do not depend on it. Although owing to the fact of its nature, it is not suitable for storing long-term data. Cinder replaced nova-volume, it supports iSCSI, NFS, Ceph, GlusterFS. Beforehand Cinder was just a gateway between the storage, that was directly connected to the controller node and mounted on an instance through tgt or iSCSi target. From the release of Havana, Cinder has its own driver for processing tasks on the storage. An image can be created from the volume. Object storage is suitable for middle-term and long-term storage, since it enables data sharing and is always available (not attached to the operating system as block storage). Keystone is an identity service, used for user/tenant authentication and authorization. It registers all virtual instances, services and users. It communicates with all other OpenStack services. For the back-end Keystone uses an sql database. Nova is installed on compute nodes, it is a hypervisor, hosting virtual instances of the private cloud. Glance is serving images to Nova, Quantum provides network connectivity between the services and for the cloud instances. As for Quantum, network as a service, several implementations are possible, among them most common are Cisco UCS plugin, Linux Bridge plugin and openVSwitch plugin. Private cloud network topology is supposed to be divided into 3 separate networks: management network for all main OpenStack components, services, service network for the traffic between instances and network services and public network for floatings IP-s. Depending on the existing infrastructure and needs, each organization or enterprise can add other networks, those three are the base minimum.

Advanced Resource Connector in the Cloud

8

8.1 ARC-CC

The Advanced Resource Connector (ARC) [43] is an open source, lightweight, portable, scalable grid service middleware solution for distributed computing resources use. ARC provides integration and management of computing resources and storage resources. It facilitates large data throughput workflows with data staging and caching implemented at the middleware layer. It is currently geared towards computing clusters managed by a batch system and remote-access enabled storage facilities. Due to its lightweight implementation, high performance and no requirements for deployment on individual compute nodes, ARC is suitable for use in versatile environments, including HPC centers and in the IaaS cloud. ARC provides a unified layer for discovery, brokering, and control of these resources via a secure common layer compatible with current grid API layers. ARC currently provides grid technologies that enable resource providers to share and federate computing and storage resources distributed across different administrative and application domains and is used to create grid infrastructures of various scope and complexity, from campus to national grids, composed from different types of resources, from desktop machines and workstations to big clusters and HPC centres. ARC with is ability to enable cross-organizational distributed computing and its strong commitment to open standards and interoperability is in a good position to enable users and research communities to access new resources and new platforms using existing interfaces and standards and to provide a way to seamlessly evolve its interfaces to enable existing users and communities to take full advantage of new resources and technologies as they become available. User communities that use ARC directly don't want to change anything when using cloud infrastructure. Workflow management could keep the same interfaces. Cloud-based resources lack support for distributed data-intensive batch processing. HPC centers lack support for data management, federated identity management and externally managed authorization. HPC centers lack cross-domain and cross-organizational flexibility. Small and large batch-oriented, possibly massively parallel, vector (GPGPU) or MPI -enabled payloads could be processed in the same, established manner on existing (EGI, PRACE, national) and future supercomputer or IaaS-based infrastructures and interfaces reused for the batch-computing components in more complex computing tasks deployed in the near future.

We used ARC compute element as a junction of a hybrid grid and cloud architecture. Our integration model, called ARC-CC, is a deployment of an ARC Cluster on demand

in the IaaS Cloud. A user can deploy her own ARC cluster in the cloud and submit grid jobs to it. This solves problems of many scientific domains, since it enables job submission to the cloud via ARC interface and more importantly does not require any customizations of the scientific workload. It is also cloud platform independent, as it can be deployed on every public and private cloud. The approach is user-oriented, offers additional resources to the users for their job submission when conventional resources are full – cloud bursting. The solution is applicable to private usage by installing private temporary CA or for existing projects and VO-s by using EugridPMA [131] certificates. It promotes the usage of the cloud in a well-known grid way. ARC-CC's main components include a virtual ARC-CE and worker nodes, built from custom virtual machine images with Packer [136], private Certificate Authority, HTCondor LRMS [25] and VOMS plugin for private OpenStack cloud. Wheras using ARC-CE, important features of the system are already implemented in the service.

Definition 7: Cloud bursting origins in hybrid clouds and implicates the burst of the private cloud to the public cloud when requests arise. In general sense cloud bursting means spreading resources to the public cloud.

8.1.1 ARC-CE

The Compute Element (CE) offers functionalities to move data files in and out of the grid cluster. It manages the runtime environments, task execution and monitoring of the task. It involves a scheduling software with queues (HTCondor in our case), serves users and includes a short-term storage for the tasks. The main components of ARC-CE are shown in Fig. 8.1 [43]. A job is submitted to ARC-CE via ARC client. A description file in xrsl [133]contains the information about the required resources and is sent together with the program and execution scripts via gridFTP to the ARC-CE. Grid manager, called A-REX parses the job description and sends it to the local resource manager (LRMS). The job is registered in the information system and sent to the worker node, where it is executed. Download and upload scripts take care of the input and output file transfer.

While distributed infrastructures are built of heterogeneous resources, delivering a proper environment for an application to run is a challenging task. Two scenarios are used to provide a proper environment, in the first one the required software is sent with

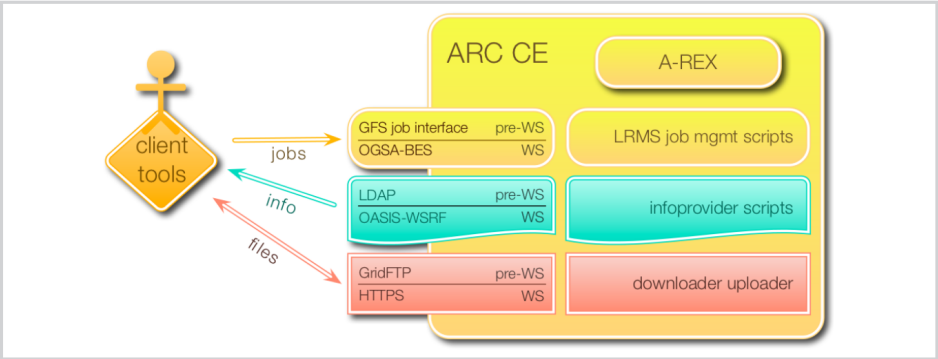


Figure 8.1

Components of the ARC Compute Element

the task via user interface (client), while the client provides installation and configuration of the software on the CE. The workflow in this scenario is as follows: first, the client searches for an appropriate CE in the information system, where all the services are registered. Task is submitted with the description about the required resources and software. Input data is transferred to the CE. On the CE, environment is set and the task sent to the scheduler. After the job in finished, CE transfers the results to the location, specified in the job description. Results or task logs can be downloaded by the client.
The second scenario assumes an existence of a pre-installed runtime environment on the grid. RTE provides pre- and post-installed script for the task. The workflow in this scenario is the same as the first one, the only change is the job description, which contains the information about which RTE to use for the task execution. In our virtual cluster solution, we can follow both scenarios. For frequent use cases, setting up an RTE seems more reasonable. RTE-s are more convenient also when a program is not lightweight.

8.1.2 HTCondor

The choice of a batch system is very important. It should be able to scale easily to a large number of nodes. Although we run SLURM as a batch system in our current grid setup, we have switched to HTCondor for our virtual cluster solution. HTCondor does not need a list of worker nodes in the configuration, opposite to other hardwired batch systems (SLURM has worker nodes list in slurm.conf, torque in server_priv/nodes). In the cloud, resources are appearing and disappearing on demand, therefore maintaining such a list would be difficult. HTCondor is also based on the server-client concept, but it is the client that sends information about being online and joins the cluster. *Startd* communicates with *collectd*, which is the main process. Collector maintains the details about

resources. When *startd* stops sending updates, it is removed from the cluster within a time interval that is set in the configuration. HTCondor can manage resources in the cloud and can maintain dynamic resources.

Resource management in the hybrid infrastructure includes the following components: Cloud provisioning , Image storage , Service discovery, Security, User Management, Information system and Data handling.

8.1.3 Cloud VM provisioning

This component is responsible for cloud resource management. It is a complex service that detects high peak demand and triggers cloud instances formation. By parsing a job description, an appropriate type of instance has to be chosen, resources allocated and job transferred to the cloud. After the job is finished, the VM should be reused for another job with the same software and hardware requirements or it should be terminated. It should be pointed out that before VM provisioning, a billing agreement needs to be established with the IaaS provider. In the ARC-CC model, cloud provisioning is done manually by a post install script. Virtual machines are terminated when the jobs are finished.
To summarize, VM provisioning deals with the following functions:

- start VM,
- stop VM,
- detect high demand
- choose OS image,
- choose security group,
- attach SSH-keys for the access to the VM.

Security group settings for our integration model require network security configuration. NFS, Condor, gridftp, ssh and other ports should be opened for grid and parallel job execution. In both, public and private cloud, we have configured security group beforehand and used it when instances were created. Public cloud providers limit the number of instances a user can start simultaneously. Not all cloud providers offer the

same selection of instance types, there are different resources packed in a certain instance. Defined resources include CPU type, number of CPU cores, memory size, disk space, network performance and I/O performance. This diversity hardens resource provisioning in different public clouds [137]. Another problem is linked to rescheduling of grid tasks to the cloud. When should tasks be executed on the cloud and when is it more economical that they wait in queue and are executed on grid? Ostermann [76] proposes a calculation of the cost per unit:

$$C_t = \frac{Cost_{cloud}}{T_{grid} - T_{grid+cloud}} \tag{8.1}$$

C_t being the cost per unit, which is calculated from total costs for leasing cloud resources (Costcloud) devided by the time, which is saved by using them (T_grid is time for task execution on grid and T_cloud on the cloud). For instance, data-intensive tasks are expensive on the cloud, while data needs to be transferred with the client to the storage cloud, from the storage cloud to the computing cloud. When the task is finished, results have to be transferred again. Data and network traffic is expensive, in this case it presumably better to wait for free resources on the grid. In our deployment, we have skipped the VM provisioning automation, our setup included a script that started a optional number of instances. This step will be automated in our further work, by using Terraform [138], Libcloud [139], Cloudscheduler, VCycle [140] or a similar suitable software.

8.1.4 *Image market place*

Since no dedicated software or service is used for VM-provisioning for now, images are created beforehand and uploaded to the image repository, managed by Glance service [60] in our private OpenStack cloud. When using public clouds, images are uploaded via User portals or downloaded from the publicly available Images repository.

8.1.5 *Service discovery*

To enable the ARC cluster in the cloud, the head node needs information about the worker nodes in the cloud cluster. Since virtual instances are spawned automatically, IP addresses or hostnames are not known in advance, a service discovery tool is needed to manage communication between the instances in the virtual cluster. A discovery service has to entail important features: service directory, service registration mechanism, monitoring and connection to services. Among available solutions, such as Cubby [141],

Zookeeper [142], etcd [143], we have chosen Consul[144]. Using a key-value datastore of services, Consul also enables monitoring, failure detection and locks. It uses a basic HTTP API for reading values and enables different consistency level (opposed to the basic server-client model) – including locks and leader election. The main advantage of this service is, that it enables clusters federation and it has a service catalog. In ARC-CC, service discovery is needed for the configuration of NFS on the head node (export file is automatically updated with allowed IP addresses) and the update of hosts file with newly spawned cloud instances. We have chosen HTCondor for the batch system, since no reconfiguration is needed on the headnode, when new nodes join the cluster. Clients are automatically added when they appear available.

8.1.6 Security and user management

In the cloud, user management differs conceptually from user management in grid. While cloud follows the simplicity and user-friendly approach, grid prioritizes higher level of security and privacy. In grid VO user authentication is used, based on x509 certificates. VO defines security policy and is used across multiple domains. A CA grants user certificates that are furthermore used for generating proxies and job submission, SSL is used for transfer and x509 proxy certificate for grid security infrastructure (GSI). A user in private and public cloud is usually authenticated by using his username and password, as for example in OpenStack or Google Engine, sometimes x509 pairs are used, for instance in Amazon EC2 or Eucalyptus. Some cloud providers see a solution in single-sign-on identities (SSO), federated identities with a service provider (idP) [145], which is still using a username and password, but these are valid across multiple services.
Authentication and authorization mechanisms are more complex in grid, but they enable collaboration of multiple users on multiple clusters. We wanted to adopt the same practice in the cloud. OpenStack uses user-tenant approach for authentication, so we decided to use Keystone VOMS module, which enables user mapping from VOMS to keystone and was developed for the EGI Federated Cloud [146]. It enables mapping of VOMS attributes to local users on the system. Keystone, identity service used in OpenStack, is a WSGI application and is deployed behind Apache. Fig. 8.2 shows the authorization procedure [15]: the web server verifies x509 user proxy, CA-s and CRL-s. WSGI filter maps VOMS attributes to OpenStack tenants and users. User creates a proxy, connects to https, his x509 proxy is validated and his attributes verified in VOMS server. First the VO is mapped to a local tenant. If a user is authorized, his VOMS attributes

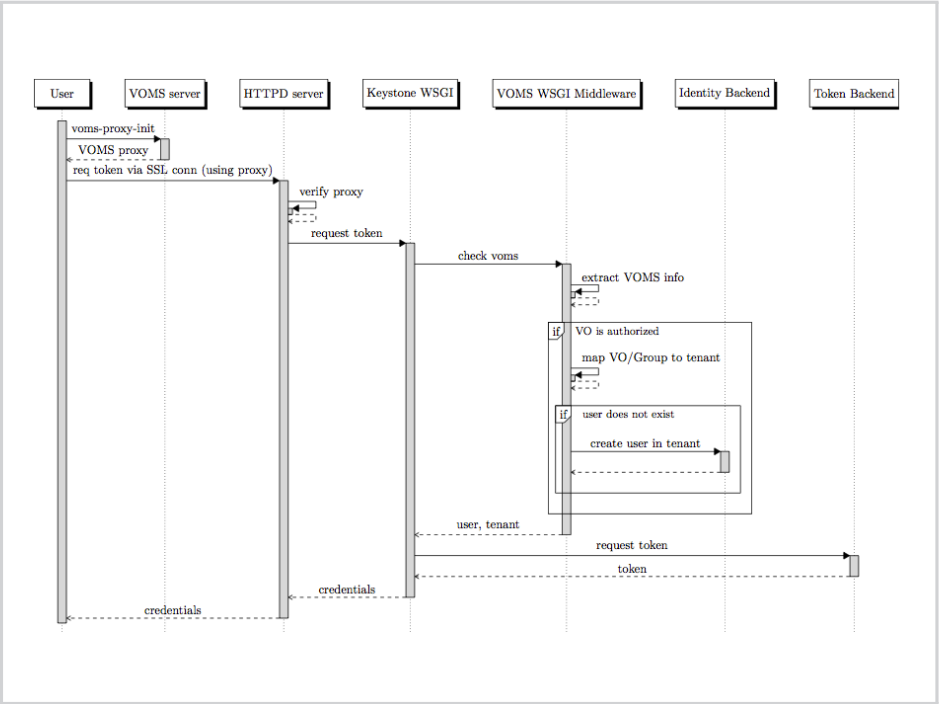


Figure 8.2

VOMS-enabled identity in Keystone.

are mapped to a local user. If a local user does not exist, a new user is created under the appropriate tenant (VO). User credentials are sent to the middleware, a token is created and used in Keystone [15]. In fact authentication in Keystone is a two step mechanism. In the first step a user initiates authentication against Keystone and a token in issued. In the second part this token is used for authentication for all other OpenStack services. The token has limited validity and is only valid within one tenant (even though a user can be a member of multiple tenants). With a VOMS Keystone module [15], users can do VM provisioning on their own.

In ARC-CC VM provisioning is done by using a dedicated script that has to be executed manually, so authentication with certificates can be used and authorization via VOMS, which is a well-established practice in grid. We support two possible authentication mechanisms. The first one uses a private Certificate Authority (CA), where a self-generated CA is used for creating and signing host and user certificates in ARC-CC. Private CA is destined to users that do not exhibit a certificate issued by a national EugridPMA CA. A private CA also generates a certificate revocation list (CRL). The

other possibility makes use of EugridPMA certificate as in grid, which enables transparent inclusion of virtual grid resources to the information system. In this case a host and user certificate are required to be included in the installation scripts. Though the whole process of resources registration and job submission is simplified, it is not trivial to obtain those certificates, while a user needs to be authorized and the procedure usually takes a few days. A certificate can be requested in advance, but it would require a dynamic DNS usage. Otherwise a DNS entry would be required for the headnode of the virtual cluster.

8.1.7 Data handling

Data handling is important on grid clusters, where computational tasks require access to big datasets, have large input files and produce results that have to be permanently stored. In ARC-CE, storage is used for caching the input data, for runtime environments, for working directories, for saving job sessions and for storing results. Since cloud was developed for services, it contains a lot of storage capacities, but their usage is not user-oriented, therefore big data transfers are expensive and can be slow. Because of different functionalities, three types of storage were developed in the cloud: block storage for local virtual machine storage, object storage for external data storage and image storage for storing virtual machine images. To be able to run grid jobs in the cloud, storage has to be treated with care. Storage is usually separated from the compute nodes. When jobs require minimal input and output data, local storage of the virtual machine is sufficient. A user can transfer the data to the cloud or from the cloud via ARC client. For more complex cases, data transfer is a bottleneck, since transferring data and bandwidth are charged by the public cloud providers and network capabilities are limited. Also application environment is a problem in the cloud. In grid two approaches are used: sending the required software with the job to the compute node or preinstalled software in the form of RTE (pre- and post-install scripts for batch systems) or CVMFS [92]. In the cloud both can be used, but the first is suitable for small and simple programs and data. Data should already be available in the cloud, access to it needs to be defined in the customization phase of the VM. To achieve better transfer rates, storage should be dispersed on different geographical locations and mounted to the cloud. When data is already in a storage, provided by a public cloud, an access to it should be configured in the VM customization phase. The same storage can be used for input and output data.
OpenStack has permanent and ephemeral storage. Permanent storage solutions include object and block storage, ephemeral storage is not permanent, it effectively disappears

when a virtual machine is terminated. In our OpenStack setup we use both, Cinder as block storage and Swift as object storage. For an image storage, Glance is used [14]. Block storage provides volumes for compute nodes and is suitable for input/output data and caching, since application do not depend on it. Although because of its nature, it is not suitable for storing long-term data. Object storage is suitable for middle-term and long-term storage, since it enables data sharing and is always available (not attached to the operating system as block storage). Data can be shared with other members of the VO. Block storage is used for working directory, since data is written and read to/from the disk all the time during job execution. Block storage is also used for caching and storing input files.

For our setup, we used Cinder for caching and input files and object storage for long-term storage in OpenStack. In public cloud we allocated 40 GB of local virtual machine storage for job execution and did not test other storage possibilities (for CVMFS repositories 20 GB suffices, the rest is used for the operating system and job working directory). In a public cloud the choice of the size of the virtual machine depends on the provider's options.

8.1.8 Information system

Compared to the grid, CE discovery is not done with the information system, as the head node on the IaaS cloud is already known. All RTE-s, users, services, resources and jobs are registered in the information system. A batch system checks available resources and reports them to the information system. The whole process is transparent to the user. Jobs are also registered in the information system, so a user can query the job status. When cloud is integrated in the grid infrastructure, for instance when integrating grid and public or hybrid cloud, resource description can represent a problem, because different cloud providers do not share the same definitions for their virtual machines. It would be necessary to maintain a database of different VM types, which would be a painstaking manual work, so that VM would be launched based on resources requirements in the job description. Registering service and jobs in the information system was one of the reasons a virtualization of a grid cluster was chosen, not an extension of the existing grid. While a virtual cluster is built, information system is already a part of the cluster and does not need to be modified, nor pre-configured instances in the database.

8.1.9 Summary of integration model

A packer template with a kickstart file and post-install bash scripts is used for creating virtual machines in the cloud. Optional images can be used, proprietary images from public cloud providers can be modified, optional software can be required, optional packages can be installed. First a head node is installed with a pre-installed script. Other available cloud resources are used for worker nodes installation. The head node runs ARC and HTCondor (information services, batch system, grid-manager, gridftp service and data storage). Worker nodes join the cluster using the Consul service discovery – IP addresses are added to the hosts file, required by the batch system, NFS exports configuration in added on the head node. All worker nodes share the same batch system configuration, containing the name of the head node, scheduler, paths to execution files etc. Certificates are installed (both options are supported – using own CA or using EUgridPMA certificates. In the latter case, a user has to add the certificate in the installation script (see Fig. 8.3).

When ARC cluster is up and running, a user can submit his jobs directly to the head node, which sends the job to the worker nodes and registers it in the information system. A user can query the status of the job in the information system. Job submission, job registration and information queries remain the same as in the grid infrastructure. The results of the job can be sent from the worker node to the head node or they can be directly copied to an external storage element, such as dCache.

8.2 Testbed

As a proof of concept, we have deployed ARC-CC on an Amazon EC2 cloud, for running simulations EC2 compute instances were used, which are suitable for CPU-intensive workload. Running the simulations in the cloud takes a lot of knowledge and time. A researcher has to prepare input data, submit simulations and transfer results when the job is finished. VM-provisioning is done in multiple steps. First a selection of an AMI virtual machine image from the AWS Market place takes place, then creating and uploading an SSH-key to access the instances, defining the number of instances to run and their properties, defining security group(s) that provide all necessary network and firewall configuration and at the end the selection of an instance type and launching the instance. After the instance is running, all necessary software needs to be installed, NFS environment enabled for users' home directory and passwordless access among the

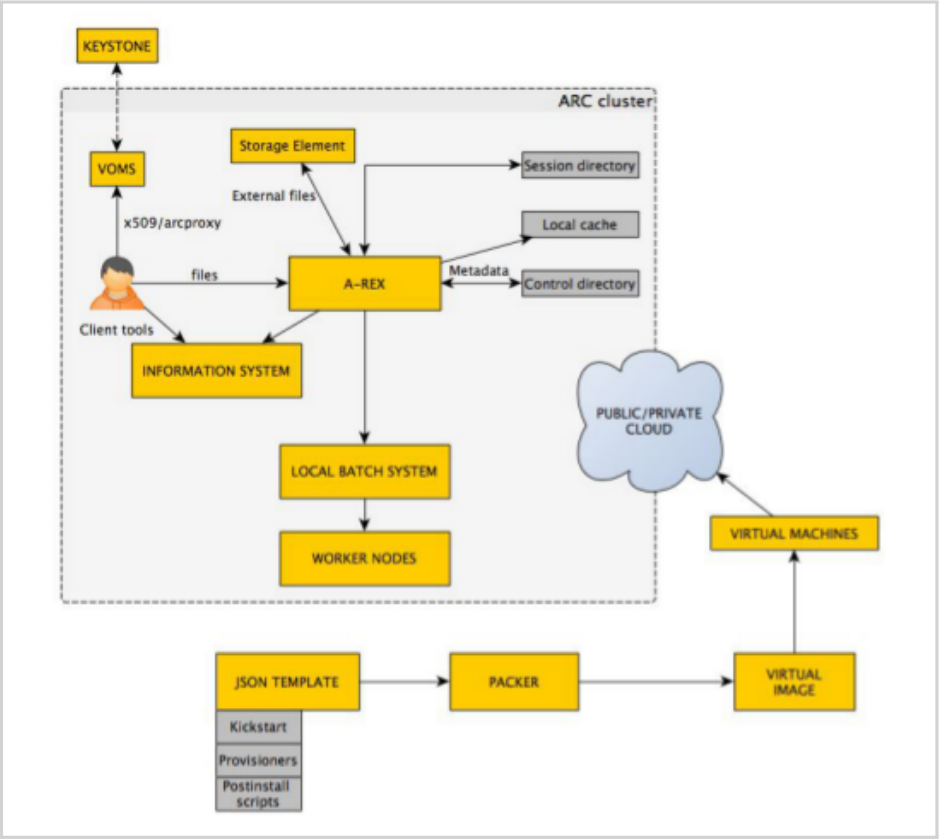


Figure 8.3

ARC-CC: ARC c
the cloud

machines in the virtual cluster by ssh keys. SElinux and iptables, enabled by default on CentOS images, need to be disabled or properly configured. In ARC-CC we included all the virtual cluster configuration with users' management, software installation and monitoring. A researcher transfers input data via ARC client, monitors his jobs and transfers results to his workstation when the job is finished.

Evaluation of ARC-CC

9

Clouds propagate fast, cheap and reliable lease of unlimited resources, but their potential is still relatively unknown. Some initial benchmarking results are available, but are not very useful of a scientific community. Parallel application executions were tested using MPI and resulted in a low performance, due to slow network interconnect [109].
We have deployed an ARC cluster in the private OpenStack cloud, on a bare metal server and on EC2 cloud, using compute optimized instances (C4 and C3 [147]). They are more expensive, but provide high CPU performance, have faster network links and they support enhanced networking, in which clustering is supported. Virtual cluster of virtual machines with low latency can be deployed. Single node testing was performed on a C4-xlarge instance with Xeon E5-2666 v3 Haswell processor. This instance type includes 4 CPU-s, 8 GB or RAM and 750Mbps dedicated Amazon Elastic Block Storage throughput (EBS), C3-xlarge that was used for multiple nodes testing, contains the same resources capacity, but has SSD disks instead of EBS storage and Xeon E5-2680 v2 processor. Scalability was tested for up to 24 CPU-s, while extra instances cannot be run by default. For bare metal testing, we used existing grid and cloud infrastructure, running on Xeon E5-2650 v2 nodes. Both KVM-based and LXC-based ARC-CC was analyzed. Linux containers (LXC) are lightweight compared to KVM virtual machines, as they only include applications and their dependencies, on the other hand they only enable deployment of different Linux distributions, not Unix, MacOS or Windows. KVM is more flexible from this point of view, but demands more resources and has higher operational costs.

9.1 *Initial performance analysis*

Initial performance analysis was executed on the virtual and physical (bare metal) ARC cluster, using a Sysbench CPU and memory benchmark [148] and Phoronix Test Suite [149] for file decompression. Resource provisioning was done manually, but will be automated in our further work. Cloud provisioning is a complex process, whereas it is not only starting and terminating virtual machines, but includes security definitions for virtual machines, user demand detection, porting of applications and tasks to the cloud etc. Memory tests using Sysbench showed on overhead up to 7% on the KVM-based instances (Arnes OpenStack cloud), on the LXC (Arnes OpenStack cloud) the performance was comparable to the bare metal server (Arnes grid cluster). On Amazon EC2 results depend notably on the instance type. Fig. 9.1 shows that general purpose instances have almost 40% of memory overhead, tested on T2 EC2 instances [147], whereas com-

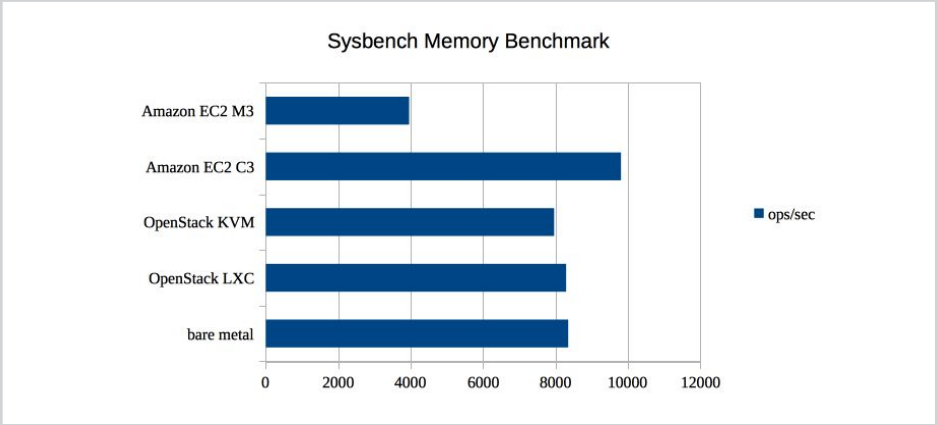


Figure 9.1

Memory performance on private and public clouds in comparison with bare metal servers.

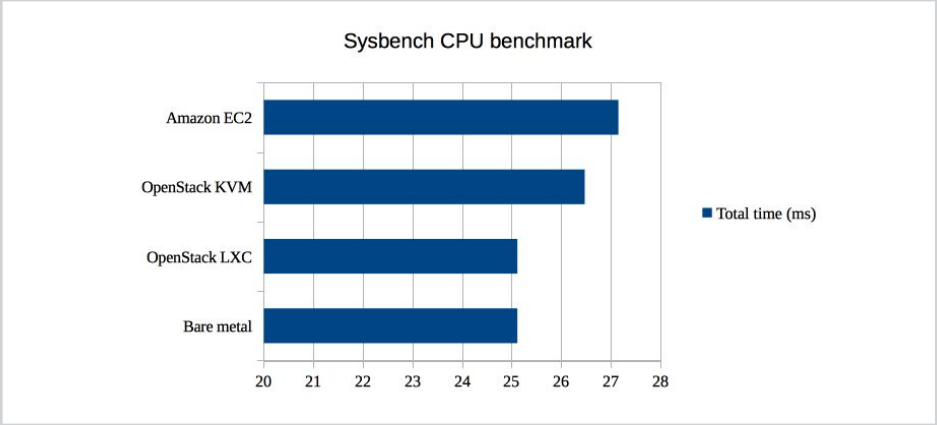


Figure 9.2

CPU performance test: maximum prime number checked in CPU Sysbench test. Results are in ms, lower number means better performance.

pute instances perform even better than our bare metal servers.

CPU-performance (see Fig. 9.2) was tested by calculating 2000 maximum prime numbers. There was no performance penalty on a LXC-based instances at Arnes, 8% lower performance was measured on a KVM-based ARC cluster at Arnes, 8% on Amazon EC2 compute instance and 40% lower on the Amazon EC2 using general purpose instances.

We have performed some tests for a decompression of an 80 MB file, using Phoronix test suite. A decompression took 12 seconds on the LXC and bare metal clusters at Arnes and 16 seconds on the KVM-based ARC cluster at Arnes and Amazon public cloud (see Fig. 9.3).

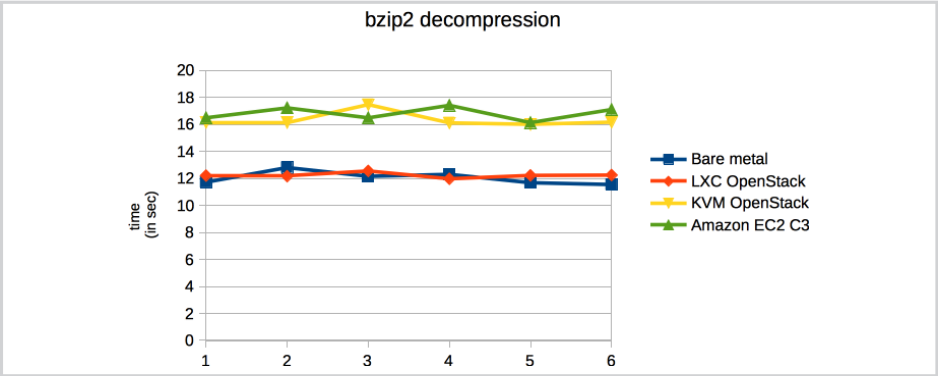


Figure 9.3

Decompression o bzip2 file.

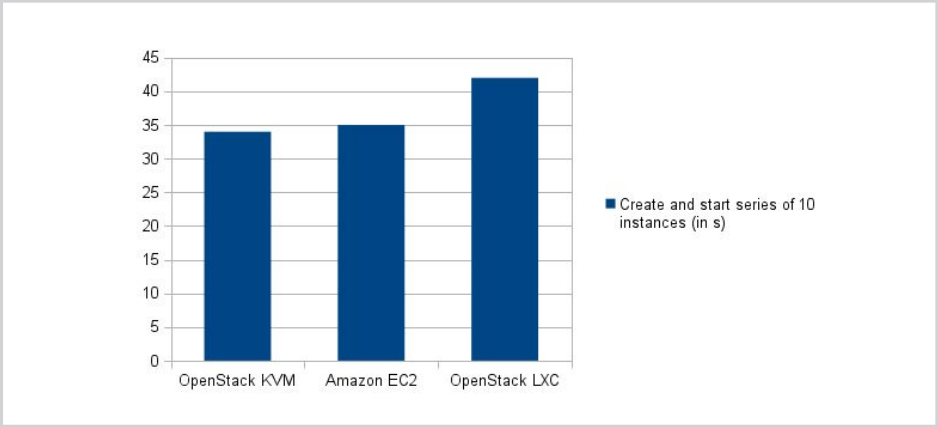
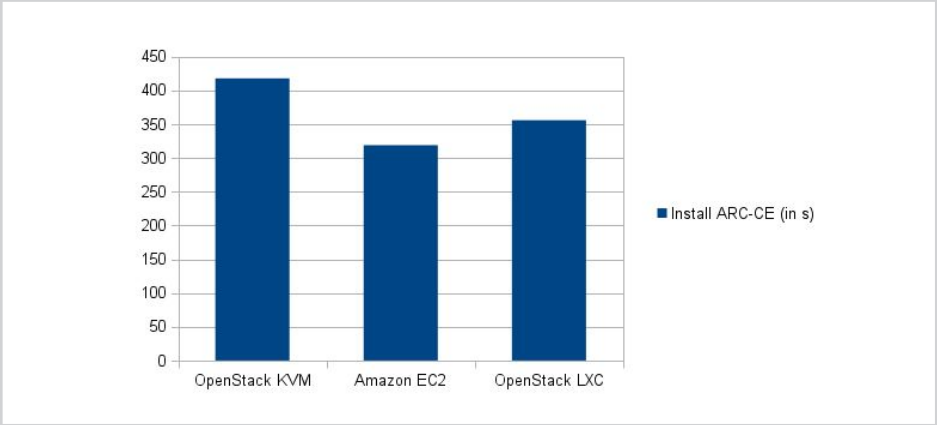


Figure 9.4

Time used for creating and starting a series of 10 cloud instances.

9.2 Building a grid cluster in the cloud

Large computing simulations require running software stacks on HPC systems, that have fast CPU-s and ensure low latency. While in house HPC clusters are expensive to maintain and financial capacities are often exceeded, we would like to create virtual clusters in the public clouds. We have measured the time to run a series of 10 cloud instances (see Fig. 9.4) on the OpenStack and EC2 cloud. Instances are up and running relatively fast. Provisioning on public and private clouds is comparable. Extending the virtual cluster with additional resources is easy and fast. Provisioned instances usually include basic software (minimal install), in ARC-CC other required software is installed by post-install scripts that are executed by Packer. Time to build a virtual cluster depends on the repositories used, virtual instance location, mirrors speed, network throughput and



9.5
uild a grid cluster
d.

the required packages to install. Condor mirrors may be faster on Amazon EC2, since our Amazon cloud instances were running in the USA as well. To increase speed, some packages could be included in the image of the virtual machine. Deployment of ARC-CC in the cloud is slightly faster on the public cloud. Recently, cloud providers have offered HPC-clusters to academia, they promise seemless access to additional resources but users without technical knowledge of the underlying software and programming skills have difficulties using it [150], which was also one of the reasons to deploy virtual ARC cluster. Users can run simulations without changing the workload and submission techniques. Only different compute element is used for their job submission. Virtual ARC cluster, ARC-CC, is built within a few minutes (Fig. 9.5).

9.3 Evaluation of ARC-CC for the scientific computing

At Academic and Research Network of Slovenia, a 4000-core HPC cluster is available for researchers and students. Apart from high energy physics, most users come from computational chemistry and biochemistry. We have chosen NAMD [151] for our performance analysis, since this program is widely used among our users. NAMD is a parallel molecular dynamics program, used in life science research projects, in computational biochemistry, chemistry and biology and is highly scalable, particularly for simulations of hundred thousand atoms or more. Our benchmark performs 500 steps, which is from 20 to 25 cycles. At some point adding new resources no longer brings any performance improvement, which needs to be tested on a real workload. Our cluster has 98% occupancy. In times of peak demand, jobs can wait in queues for up to 4-5 days. In this

case our researchers could benefit from public cloud resources. We performed NAMD benchmarking analysis on a physical (bare metal) and virtual ARC cluster (using OpenStack with KVM hypervisor and Amazon EC2). We used ApoA1 benchmark [152] that has been the standard NAMD cross-platform benchmark for years. NAMD releases are available also for GPU-s and CUDA, Xeon Phi, OpenMPI with Infiniband, etc. No suitable-for-all-environments version is available. We have used NAMD multicore on single nodes and NAMD Linux Ethernet for multiple nodes.

9.4 Scalability efficiency of NAMD on single node

We have created instances via AWS Management console. In the provisioning process, VM image is chosen, security group defined (which ports to open) and instances launched. After installing the virtual ARC cluster, NAMD software needs to be copied on the servers. Users and execution enivronment are created by Packer post install scripts. Running NAMD on a single node showed similar results than running a simulation on multiple nodes with the same resources sizes. Our tests on a single node showed that CPU scaling efficiency for 4 CPU-s is 95% on a physical machine, 94.5% on a KVM-based instance and 92% on Amazon EC2 instance. For 12 CPU-s the efficiency was 79.2% on a physical machine, 62% on a KVM-based cloud instance and 45.3% on Amazon cluster. Since the virtual instance on Amazon EC2 had 8 CPU cores, benchmarking results for 12 CPU-s on a single node on Amazon are missing. Memory consumption was similar on virtual and physical ARC cluster (see Fig. 9.6) Up to 2% of performance can be gained by enabling CPU affinity for the simulation. Interestingly the wallclock time increases with CPU affinity enabled, memory consumption as well, but the overall performance is better. Efficiency would be better with low-latencies, for example by using fast infiniband networks. Performance is better when using compilers and drivers that correspond to our hardware (e.g. Intel compiler or proprietary infinband drivers).

Scalability is efficient, whereas adding new CPU-s to the simulation lowers the overall performance and time spent for finishing a single step of the simulation (see Fig. 9.7 and Fig. 9.8) Results on the OpenStack private cloud, built on the top of KVM hypervisor, show that overall performance is better on the private cloud (see Fig. 9.9. Also wallclock time decreases when adding CPU-s to the simulation, as shown on the (see Fig. 9.10).

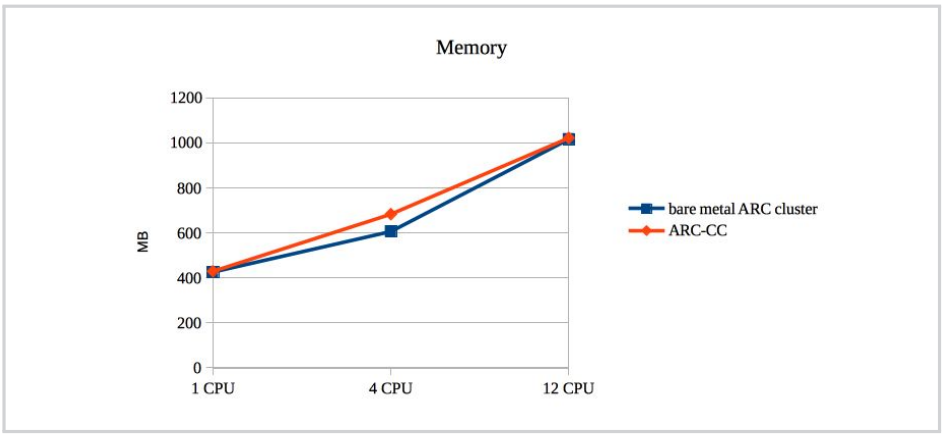
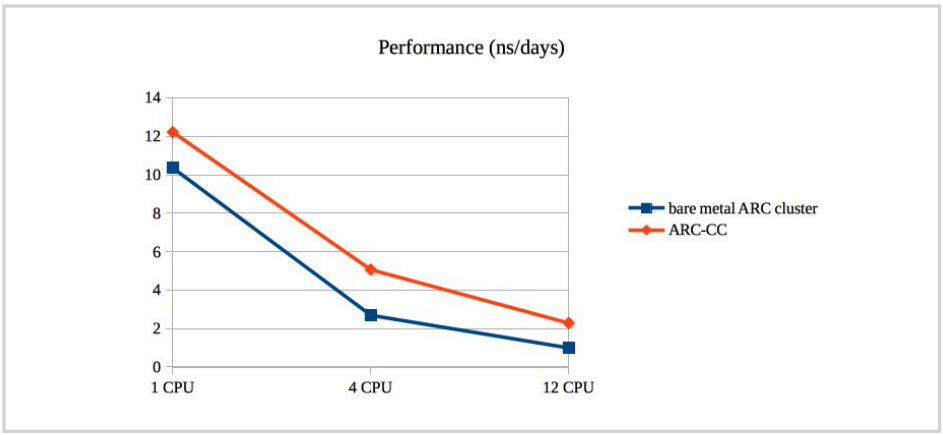


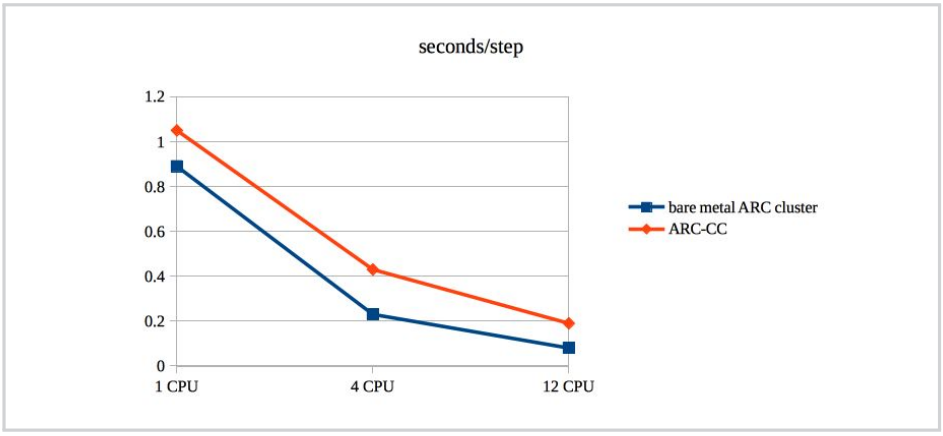
Figure 9.6

Memory consumption when running NAMD on a single node, using KVM, Amazon EC2 and bare metal servers.



9.7

'erall perfor-
ı single node
C-CC at Amazon
ietal ARC cluster
: signifies ns per
of computation
'ed per nanosec-
ulation. The
ber means better
ce.



9.8

'nchmark with
'wing how many
ent for 1 step of
'ion on a single
n bare metal
'r and ARC-CC.

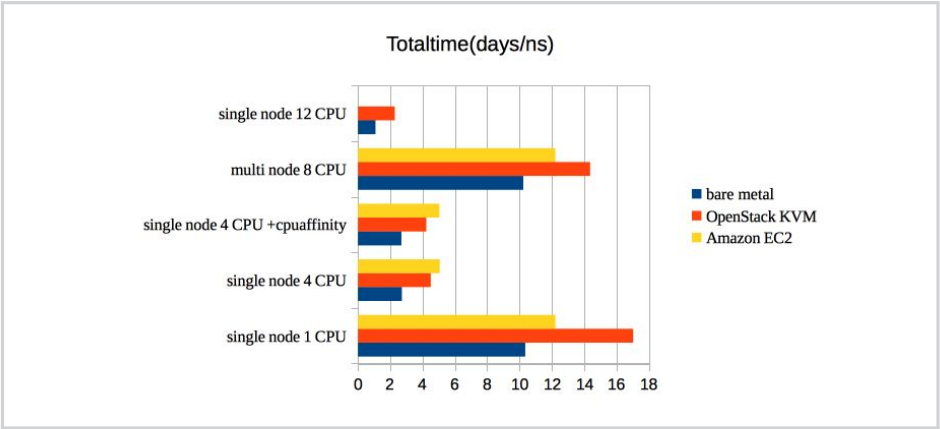


Figure 9.9

NAMD overall performance of a simulation, running on a single node,on ARC-CC, built on OpenStack private cloud, ARC-CC, built on Amazon EC2 and bare metal ARC cluster.

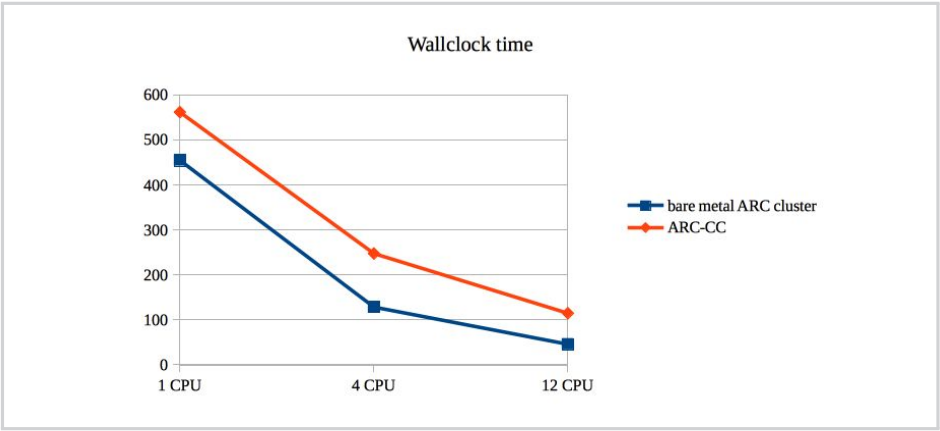


Figure 9.10

Wallclock time fo
simulations on a
machine within A
EC2 and bare me
cluster.

Definition 8: Wall clock time, as the name suggests, is the time measured from the start to the end of the process as if it was measured by a clock on the wall. Wall time or wall clock time is the overall performance time, the elapsed time, the time between the start and the end of a process. Other processes, running on the same physical hardware, effect the wall clock time.

Definition 9: CPU time or process time is the time spent by a CPU for a process. We distinguish between user and system CPU time, system CPU time being the time used for servicing system calls and user CPU time as the time used for non-kernel operations.

In scientific computations both CPU time and wall clock times are important. The duration of the calculation is defined by the CPU time used for a process. Wall clock time gives us better evaluation of the overall performance and it gives the information about the time waiting for I/O operations etc.

In virtual environments multiple tenants access the same hardware resources, therefore information on the total time for a task is very important and shows how or if other tenants influence the performance of our workload.

9.5 Scalability efficiency of NAMD on multiple nodes

Parallel environment in NAMD can be used with Charm++ or MPI libraries [153]. When running a program with less than 2 processors, processors are mapped by core, when running with more than 2 processors, they are mapped by socket. We performed the tests using charm++ via ARC.
There has been an inconsistency performing the tests on Amazon public cloud. In the warm up phase of the instance, performance results are much better. Performance of NAMD varied for up to 15% - reasons lay in Amazon EC2 internals, we presume that CPU time slice sharing is not equal among the neighbors on the same physical hardware and due to different user load, results on the cloud depend a lot on the ”noisy neighbors”. The execution time of a program cannot be predicted precisely. To prevent jobs from being killed, more resources are assigned for a job and cloud resources utilization cannot be optimal.

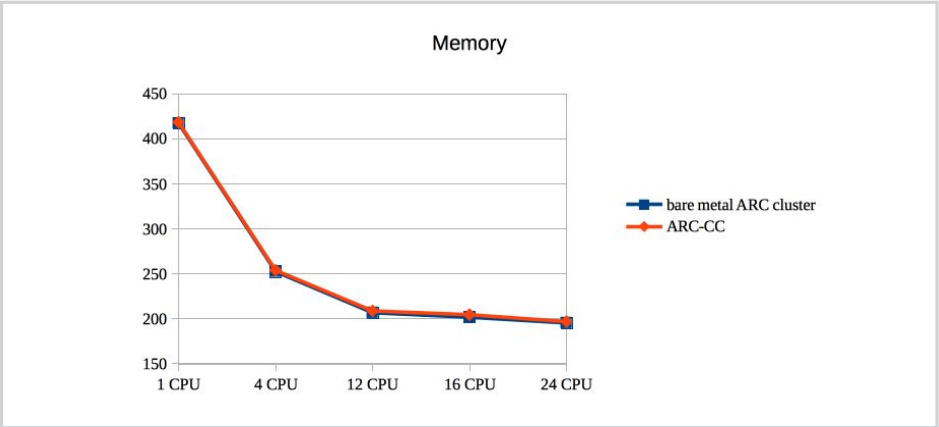


Figure 9.11

Total memory consumption by CPU when running NAMD on multiple nodes within ARC-CC on Amazon EC2.

9.5.1 Memory overhead with NAMD simulations

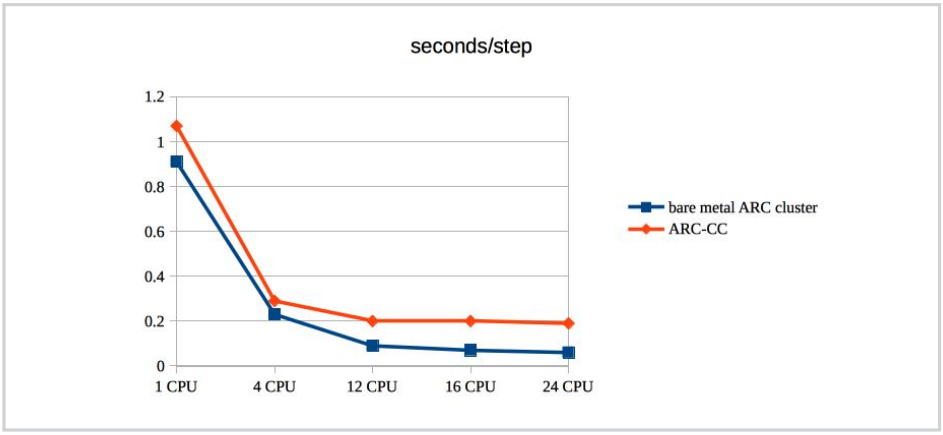
Memory overhead was very low, up to 1% as shown in Fig. 9.11. NAMD is not very demanding concerning memory usage, while it consumes around 500 MB of memory per core on average.

9.5.2 NAMD overall performance in the ARC-CC

Overall performance of NAMD simulations is lower in the virtual cluster due to virtualization itself. If multiple tenants share the same system, CPU cache has lower ratio, CPU interrupts are present from other workloads, disk is used for multiple tasks etc.
Based on our results, we consider that the program scales efficiently in the virtual cluster. As shown in the Fig. 9.12, time used for one step in the simulation decreases significantly when adding new CPU-s for the program. CPU scaling efficiency was calculated with the following equation:

$$e(n) = \frac{t_s}{n * t_n} \tag{9.1}$$

e is efficiency, n is the number of processors running the simulation, t_s is the execution time running on one processor and t_n is the execution time running on n processors. The guideline is that parallel jobs should scale to at least 70% for efficient scalability. It may be the case that adding processors would slow the actual over-all performance (wall-time) of the job, as shown in Fig. 9.13. More reliable benchmarks should be done with actual user jobs. Although performance tests on virtual clusters vary and virtual instances coexist with other programs on the same system, performance results on the virtual ARC



9.12

nds) per step
ing NAMD
s.

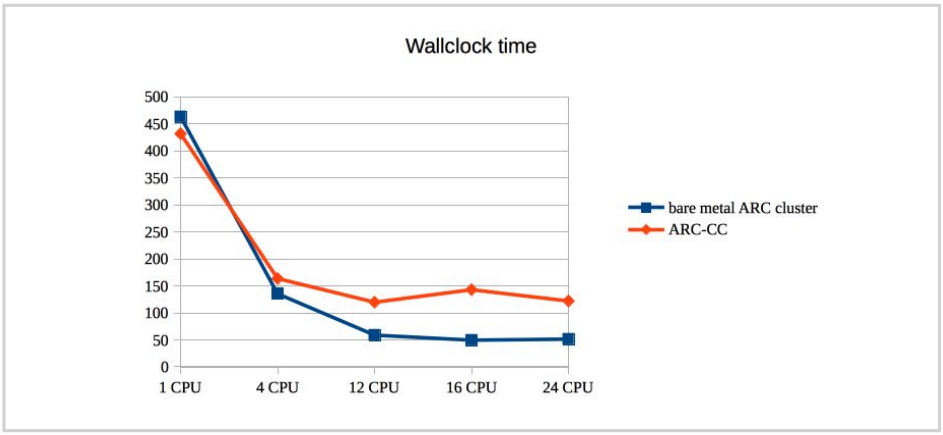


Figure 9.13

Wallclock time of NAMD simulations.

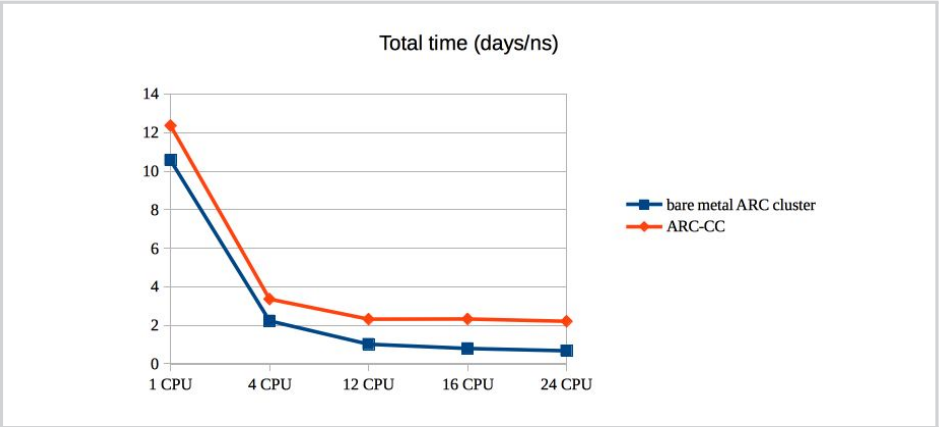


Figure 9.14
Totaltime used fo
simulations.

cluster showed promising results. All tested environments exhibited good scalability and report faster simulations every time processor count is higher (see Fig. 9.14).

9.6 Performance comparison

As shown on Fig. 9.15 overall performance is increasing by adding new CPU-cores, scalability of the simulations is efficient in all environments. While using different CPU-s for single node testing and multi node testing, results differ already on bare metal ARC cluster. Overall performance is worse on the virtual ARC cluster, the performance penalty is apporoximately 15%.

Monte Carlo simulations, running on public clouds, achieved the same results, performance penalty altered between 5 and 15% depending on the applications that ran in the cloud [33]. Testing ARC-CC on Amazon EC2 for NAMD simulations showed the expected performance penalty, discussed already in the previous chapters. Benchmark testing showed some other constraints of the cloud usage. Seemingly infinite resources turn out to be limited as well, limitations are present already in the VM provisioning (limit of maximum instances one can run), availability of certain types of instances (C4 instances are not available for multiple node simulations) and a large number of acquired IP addresses. VO-support is complex to implement, since an EUgridPMA certificate is required for the headnode and the procedure of acquiring can be slow. Although a lot of effort is put into development of short-term certificates that would enable launching EGI-compatible virtual grid clusters on demand. Some Certificate Authorities enable certificate generation via AAI interface (TERENA members) and sign certificates within

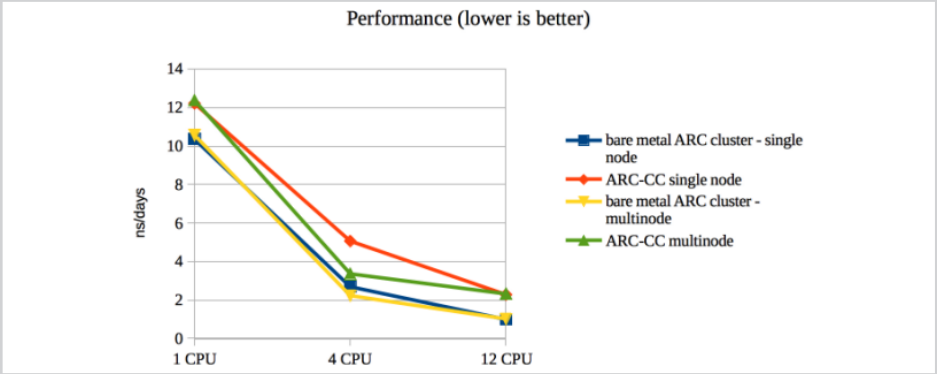


Figure 9.15

Overall performance on a bare metal ARC cluster and ARC-CC, simulations on single- and multinode.

a few minutes. For now the simplest solution is to require a certificate in advance, using a dynamic DNS. The frequency of updating should last as long as VM provisioning.

The phase of VM provisioning is very important, because different types of instances bring significant performance differencies. It is possible to run into relatively high latency problems even with low-level instances. Latency includes the time, waiting for an application to complete. Latencies are present in every step. For example network latency is influenced by the number of connections, by dropped packets, by retransmitted packets, out of order packets and errors, by DNS latency, TCP handshake latency and data transfer latency [30]. We have tested a few scenarios and conclude that the choice, which disk to use, is crucial to VM performance. When Amazon EBS storage is replaced by ephemeral storage, disk latency is lower. Some performance is gained when using SSD disks. We can decrease latencies also by caching and buffering on the system, data can be compressed before transfer, written to memory, not disk directly etc. If general instances are used for calculations, some problems will appear eventually, because they have slow disks and slow network. We should use instances that are not under the influence of other guests, that are running on the same compute node.

The integration model ARC-CC is adaptive and more dynamic as other available solutions. It can be used in different setups, for different research projects and types of workload. It adapts well to users' requirements, while execution environments are included in the VM images and can be modified as needed, a user can use well-established interfaces and can easily monitor available resources and the status of the submitted jobs. The flexibility of the model does not bring a lower security level. By using ARC middleware as the basis, including x509 certificates for authentication and VOMS-service for autho-

rization, a high level of security and privacy is provided for the user.
ARC-CC grid and cloud integration model corresponds well to the defined objectives in the beginning for this work. We have shown that the integration of two non-interoperable architectures is possible, that applications can be migrated from one architecture to another without changing the calculations and their execution methods. Even if lower performance was measured, 10 or 20% of performance penalty is still better than waiting for days in queues for on-premises or grid resources to become available. The solution can be easily deployed both in private and public clouds and is suitable to be used in times of peak demand.

Conclusion

10

The technical convergence and interoperability of commercial and private clouds with grid are among the main goals of many current grid projects. In our work we discussed the benefits of grid and cloud integration and concluded that the main advantages would be in an easy migration of applications from the grid to the cloud, lightening the operational costs, better access to available resources, customizable execution environments, elasticity and scalability.

We have developed a model of grid and cloud integration and implemented the deployment of ARC cluster in the cloud, thus matching the definition of cloud bursting. Both grid and batch job submission to the grid cluster in the cloud are supported. Our integration model enables simple usage of cloud resources when grid resources are busy or unavailable. To migrate applications in the cloud, no modifications of the scientific workload are necessary, nor changes of code or submission techniques from the users' side. The approach is user-oriented, since the environment can be set within a few minutes and easily adjusted to the user's needs. ARC-CC can be used in every public or private cloud and is also applicable in our infrastructure at Arnes. Our suggested approach is applicable for all scientific workloads and disciplines. ARC-CC provides a fully automated deployment with "simple" initial configuration.

We presented other available solutions in this field, but they usually only cover a specific use case and require at least a partial manual management.

We have deployed our ARC-CC model, which includes a unified access to resources via ARC or DIRAC client. Runtime environments are translated into virtual machine images and contain all necessary software and environment settings for a specific VO. Since public cloud providers offer their pre-installed images, the system can be later adjusted by using post-install scripts. An ARC cluster is built in the cloud by using a single json configuration file and Packer builders. The solution is portable to different research fields and use cases; problems with service discovery were successfully solved by using Consul service discovery software. Complex user management and security were also properly addressed and work the same way for the user as in conventional grid. Some challenges are still present. Data intensive workload require a big data storage in the cloud. Data in the public cloud is usually an individual entity outside of the computing cloud, data transfer and network traffic are charged to the users. For more complex and data-intensive workloads also data transfer and storage problems need to be tackled.

Performance analysis was done on ARC-CC, deployed on a private and public cloud. Only simulations were executed in this experiment to avoid data related issues. Results

show that there is some CPU overhead on KVM-based virtual machines, even higher on the public cloud, it is negligible on Linux containers. Memory overhead is minimal. However performance results can vary, when using different types of instances. Performance speed depends presumably on the neighbors' workload that uses the same physical hardware. CPU efficiency in parallel executions alters significantly and hardens the precise evaluation of how long a user workload will run on virtual resources. To prevent jobs from being killed for exceeding resource limitations, some overhead needs to be included in VM provisioning, consequently resource utilization cannot be optimal. In private cloud, access to the hypervisors is possible, therefore the execution environment can be adjusted to users' workload and better resource utilization and workload performance can be provided.

Scientific workload was executed on ARC-CC. Results show that all environments exhibit good scalability and report faster simulations every time processor count is increased. Overall performance of simulations is slower on virtual resources compared to bare metal servers. Lower performance was expected due to a virtualization layer and due to the fact that multiple tenants are using the same hardware concurrently. The system CPU and memory caching does not work the same way as on a bare metal server and noisy neighbors can cause a higher overhead. There is no (little) additional overhead due to ARC-CC deployment. We conclude that results are promising and cloud infrastructure can easily be used for running simulations that require little i/o and are CPU- not data-intensive.

10.1 Future Research Directions

Our work is a step forward in using cloud resources as complementary resources to grid. This integration model can be enhanced by enabling ARC Rainbow and different versions of operating systems in the cluster. The KVM hypervisor in the private cloud could be replaced by Linux Containers which would provide a bare metal high performance system. In our further work we will include the VM provisioning component, either by reconfiguring *condor_rooster* [154] to be used for provisioning or by using Cloudscheduler or a similar service.

All future developments should answer to the needs of grid users and continue to provide a reliable infrastructure for them. Some challenges are still present. Data intensive workload require a big data storage in the cloud. Data in the public cloud is usually an individual entity outside of the computing cloud, data transfer and network traffic

are charged to the users, a decent storage performance as well. For more complex and data-intensive workloads data transfer and storage problems need to be tackled and the performance to cost evaluation automatically addressed.

While using clouds for computing tasks is still a new approach in scientific computing, it will definitely improve in the future development. The current results are promising, even though the performance is lower. But a boy cannot jump 2 meters long when he is two, but can do it when he is fifteen.

Razširjeni povzetek

A

A.1 Predstavitev problematike

Porazdeljeno računalništvo se je zadnja leta hitro razvijalo. Porazdeljenost je prisotna v različnih infrastrukturah; v gručah, gridu [1], oblaku [2] [3], podatkovnih shrambah, visokozmogljivem računanju (ang. HPC), omrežjih itn. Kompleksnost porazdeljenosti je povezana z različnimi napravami, ki do te infrastrukture dostopajo, z različnimi pravili uporabe, različno strojno opremo, formati, protokoli in omrežnimi povezavami [4]. Sočasno z razvojem porazdeljenih sistemov, so se razvijale tudi aplikacije in programska oprema, ki na takih sistemih tečejo. Visokozmogljive omrežne povezave, tehnologija infiniband in komunikacijski protokoli so zmanjšali težave s pasovno širino in zakasnitvijo med razpršenimi vozlišči in procesorji, ki sestavljajo tak porazdeljen sistem [5]. Računalništvo grid se je pojavilo v 1990. letih in sloni na isti ideji kot splet. Glavni cilj te tehnologije je souporaba podatkov, ki so shranjeni na porazdeljeni infrastrukturi in souporaba računskih in podatkovnih kapacitet. Viri so uporabniku na voljo na zahtevo [6]. Pod porazdeljenost v sistemih grid štejemo tako porazdeljenost virov, kot tudi porazdeljenost nalog, ki na gridu tečejo. Med porazdeljenost virov štejemo administracijo in upravljanje podatkovnih in računskih virov. Porazdeljeni računski viri so osnovna komponenta sistema za pošiljanje nalog. Naloge v porazdeljenem sistemu so poslane vozlišča, ki so fizično v isti računski gruči. Zahteve po računskih virih strmo naraščajo. Omrežja grid so običajno prezasedena, saj uporabnikom ponujajo na voljo le omejeno število fizičnih virov in specifična okolja za izvajanje nalog, ki omogočajo le delno spreminjanje in konfiguracijo. Grid ne zmore odgovoriti na zahteve vseh uporabnikov. Zaradi narave te tehnologije, mora uporabnik pred izvedbo eksperimenta, naloge najprej preizkusiti na vsaki gruči, na kateri bo računal in izvršljiv program ustrezno prilagoditi. Poleg možnih težav pri izvajanju nalog, je potrebno upoštevati tudi vrstni red njihovega izvajanja. Le-ta se izračuna na podlagi števila že izvedenih nalog (podatki o zgodovini), upravičenega deleža do uporabe gruče in pripadnosti virtualni organizaciji [7]. V primeru velikega povpraševanja po strojnih virih in daljših čakalnih vrst pri izvajanju nalog, bi uporabniki lahko koristili proste kapacitete v zasebnem ali javnem oblaku. Oblak bi tako uporabili kot dodatek k obstoječi infrastrukturi grid.

Oblačne rešitve predstavljajo nov pristop k znanstveni obdelavi podatkov. Oblak namreč ponuja na videz neomejene vire in je dosegljiv vsem uporabnikom. Zagotavlja vire na zahtevo, omogoča postavitev več različnih okolij za izvajanje nalog in je relativno enostaven za uporabo. Z uporabo javnega oblaka stroške vzdrževanja strojne opreme pre-

nesemo iz domače organizacije na ponudnika storitev v oblaku, s čimer zmanjšujemo lastne stroške. Računske vire v primeru neuporabe enostavno izbrišemo in ne zahtevajo nobenih stroškov. Dosegljivost virov v oblaku se dinamično povečuje ali zmanjšuje, obseg virov se prilagaja povpraševanju in so na voljo za omejeno časovno obdobje. Na drugi strani ima infrastruktura grid omejeno število strojnih virov, a so ti ves čas na voljo, četudi na njih v določenem segmentu ne tečejo naloge. Izraba strojnih virov tako ni optimalna. Kljub stalni dosegljivosti virov, se njihovo število lahko spreminja [8]. Dodatna omejitev v količini dostopnih virov je prisotna zaradi potreb eksperimenta. Če za eksperiment potrebujemo točno določeno knjižnico ali program, ki ga na gruči grid ni, se za nas število razpoložljivih virov še dodatno zmanjša.

Kljub spremembam na področju porazdeljenega računalništva in poskusom vpeljave oblaka v porazdeljeno računanje, znanstvene aplikacije še vedno slonijo na bolj ali manj tradicionali in permanentni visokozmogljivi (HPC) [9] in visoko prepustni (HTC) infrastrukturi. [10] [11]. Storitve v oblaku so nepogrešljive takrat, ko so računalniške kapacitete polne, čakalne vrste dolge, rezultate izračunov ali eksperimenta pa potrebujemo takoj. Takrat postane oblak dodatek k obstoječi infrastrukturi, govorimo o razširitvi v oblak (ang. Cloud bursting) [12]. Vključevanje oblaka v obstoječo insfrastrukturo grid pomeni velik korak naprej k znanstveni obdelavi podatkov, saj je uporabniku omogočeno koriščenje dodatnih računskih in podatkovnih virov, izboljšuje se tudi izkoriščenost strojnih virov.

Na voljo je že nekaj rešitev, ki združujejo infrastrukturo oblaka s katero drugo infrastrukturo, kot na primer spletno, paralelno ali drugo oblačno gručo, a te spremembe zahtevajo tudi prilagoditev procesa vzdrževanja, posodabljanja, nameščanja in optimizacije sistema [13].

V zadnjem času je zanimanja za širitev določene infrastrukture v oblak veliko. Cilj je uporabniku zagotoviti dostop do dodatnih virov v oblaku, ne da bi bil primoran spremeniti svoj pogram, ki se bo na infrastrukturi izvajal, in brez sprememb v postopku pošiljanja nalog. Zanekrat globalne rešitve združevanja arhitektur ni, saj se pristopi združevanja grida in oblaka močno razlikujejo. Največ raziskovalnih skupin se odloča za spreminjanje in prilagajanje programske opreme in eksperimentov na način, ki bo omogočal izvedbo na oblačni infrastrukturi. Postopek prilagoditve je zahteven, zamuden in prinaša dodatne stroške, saj ponudniki javnih oblakov uporabljajo nestandardizirane vmesnike in programsko opremo. Največkrat tako eksperiment, ki ga raziskovalec ali skupina prilagodi za izvajanje na javnem oblaku, nima možnosti prenosa delovnega toka iz enega v

drug javni oblak, temveč je skupina primorana vedno uporabiti istega ponudnika. Nekatere rešitve se zaradi navedenih težav raje poslužujejo svojih zasebnih oblakov, v katere vključujejo določene komponente omrežja grid. Npr. postavitev zasebnega oblaka s programsko opremo OpenStack [14] in sistemom overjanja preko strežnika VOMS [15], ki je sicer način overjanja v omrežju grid [15]. Tretji pristop združevanja obeh arhitektur je v razvoju združjivih uporabniških vmesnikov, ki omogočajo dostop in izvedbo nalog na obeh infrastrukturah.

Zakaj so pristopi zdrževanja tako različni? Pri izvajanju nalog znanstvene obdelave v javnih oblakih se srečujemo z najrazličnejšimi izzivi. Javni oblaki imajo različne, nezdružljive pogramske vmesnike (ang. API), različne formate, različne predloge sistemov, različne možnosti povezovanja v oblak ali upravljanja hrambe podatkov, tudi različne sisteme za popis porabe itn. Te probleme raziskovalne skupine rešujejo na različne načine, zato so tudi pristopi združevanja grida in oblaka tako različni. Slabost vseh obstoječih rešitev je ta, da nobena ni prenosljiva na druge raziskovalne projekte. Neke globalne rešitve, ki bi ustrezala vsem razikovalnim skupinam in projektom, ni. Prav tako je problem prenosljivosti rešitev na različne ponudnike javnih oblakov. Največkrat je omogočen le en sam ponudnik.

A.2 Porazdeljeno računalništvo

Porazdeljeno ali razpršeno računalništvo se je pojavilo še pred razvojem Interneta, in sicer je njegov predhodnik ARPANET ponudil prvo porazdeljeno aplikacijo že v 1970. letih, t.j. elektronsko pošto [27]. V porazdeljenih sistemih so viri razpršeni na različne lokacije. Vire lahko v sistem dodajamo in jih odstranimo, nadzor nad sistemom je decentraliziran, prav tako njegovo vzdrževanje. Okvara posamezne komponente sistema, posamezne gruče ali strežnika, ne vpliva na delovanje celotnega sistema. Vse to omogočajo dobri komunikacijski protokoli in programska oprema, ki omogoča, da heterogen in razpršen sistem za končnega uporabnika deluje kot homogena celota. Tako arhitektura grida kot arhitektura oblaka sta porazdeljeni arhitekturi, a sta namenjena različni uporabi. Omeniti je potrebno visokozmogljivo računanje, kjer so v rabi sočasne, paralelne aplikacije in naloge. Posamezna naloga tako hkrati teče na več procesorjih in strežnikih - gre za t.i. horizontalno paralelizacijo [30]. Za zagotovitev visokozmogljivega računanja je potrebno zadostiti naslednjim pogojem [5]:

- hitre medprocesorske povezave, ki vključujejo visoko pasovno širino in nizke za-

kasnitve (npr. za povezavo strežnikov uporabimo povezavo prek tehnologije infiniband),

- zmogljiva strojna oprema (hitri procesorji),
- razširljiva programska oprema, ki lahko za svoje izvajanje hkrati uporablja računske vire večih vozlišč.

Že leta 1991 je Peter Deutsch, programer, ki je razvijal programsko opremo za porazdeljene sisteme, napisal 7 zmot o porazdeljenem računalništvu [28], osmo je dodal James Gossling leta 1997:

- Omrežje je zanesljivo.
- Zakasnitev je enaka nič.
- Pasovna širina je neomejena.
- Omrežje je varno.
- Topologija se ne spreminja.
- Skrbnik omrežja je en sam.
- Cena prenosa je enaka nič.
- Omrežje je homogeno.

Te trditve so preučevali leta kasneje (npr. [29]), a avtorji jih samo še potrdili. Tudi danes so trditve resnične, a se jih morda bolj zavedamo. Ključ k razvoju aplikacij, ki tečejo na porazdeljenih sistemih, je v razumevanju omrežja. V izogib težavam, je potrebno upoštevati vseh 8 trditev.

A.3 Arhitektura grida in oblaka

A.3.1 Arhitektura grida

Omrežje grid sestavlja več heterogenih strojnih in programskih virov, ki so razpršeni na različnih fizičnih lokacijah, a s pomočjo vmesne programske opreme delujejo kot homogen navidezni superračunalnik.

Porazdeljeno računalništvo grid se je pojavilo v 1990. letih. Ključno vlogo v njegovem razvoju je odigral CERN in vzpostavitev njegovega eksperimenta Velikega Hadronskega Trkalnika (ang. LHC) [37] leta 1996. Pri projektu so namreč ugotovili, da računske kapacitete znotraj organizacije ne bodo zadostile potrebam eksperimenta. CERN je razvil idejo porazdeljenih računskih centrov in iskal rešitev, kako jih povezati med seboj, da bi lahko končnemu uporabniku sistem predstavili na enoten način in, da se uporabniku ne bi bilo potrebno obremenjevati s tem, kje so podatki, kako do njih, do katerih računskih virov je sploh upravičen in kdaj so na voljo. Leta 1998 so rešitve iskali znotraj projekta MONARC [38] in definirali hierarhijo centrov in njihove funkcije, nato so v projektih Open Data Grid, EGEE I, II in III [38] pozornost namenili razvoju vmesne programske opreme, razvili so vmesno programsko opremo gLite. V ZDA so v istem času predstavili I-Way [39] [40], predhodnika današnjega Globus Toolkit/a, ki je v ZDA še danes med najbolj uporabljeno vmesno programsko opremo. V Evropi sta se kasneje razvili še dve uporabni rešitvi, raziskovalci nordijskih dežel so razvili vmesno programsko opremo ARC [43], v Nemčiji pa vmesno programsko opremo Unicore [44]. Naštete rešitve so še danes med najbolj uporabljenimi vmesnimi programskimi opremami grid. Ker so bile spremembe vmesne pogramske opreme pogoste, se je CERN pred tem zaščitil z razvojem svojih lastnih vmesnikov in orodij, ki slonijo na porazdeljenih sistemih, kot na primer PanDA, Alien and Dirac [33].
Osnovne komponente omrežja grid so naslednje:

- sistem pošiljanja nalog z izdelano varnostno politiko (overjanje in avtorizacija)
- sistem razvrščanja nalog (ang. scheduling),
- prenos podatkov in njihovo upravljanje,
- uporaba virov.

Vse našteto je vključeno v vmesno programsko opremo, ki deluje kot vezni člen med strojnimi in programskimi viri ter virtualnimi organizacijami. Uporabniški vmesnik omogoča pošiljanje nalog na gruče, prenos podatkov, nadzor nad izvajanjem nalog in prenos rezultatov iz gruč. Na gruči se naloge razvrstijo v vrste, glede na tip naloge, delež, ki ga ima virtualna organizacija na gruči in glede na število že poslanih nalog uporabnika na gručo [3]. Razvrščanje nalog je zelo kompleksen proces, ki zahteva veliko znanja, saj lahko z njim dosežemo optimalno izkoriščenost virov. Poleg vmesne programske opreme je za

delovanje grida nujna hitra mrežna povezava in dobri ter zanesljivi protokoli za prenos podatkov (npr. gridftp).

A.3.2 Arhitektura oblaka

Oblaki so avtomatizirane, prilagodljive, nastavljive, razširljive storitve in orkestracija, ki so na voljo na zahtevo, ustrezajo različnim primerom uporabe in omogočajo maksimalno izrabo računskih virov. Oblak določajo 3 osnovne lastnosti, to so virtualizacija, storitveno usmerjena arhitektura in dodeljevenanje virov na zahtevo. Oblak je dinamicen, prilagodimo ga lahko svojemu delovnemu procesu [48], namenjen je predvsem gostovanju storitev, manj pa zahtevni znanstveni obdelavi podatkov, saj v svoji zasnovi ne upošteva velike količine vstopnih in izstopnih podatkov ter potrebe po hitrejših prenosih [6]. Računalništvo v oblaku sloni na poslovnem modelu, kjer uporabnik najame virtualne vire in jih po končani uporabi plača. Uporabnik tako ni obremenjen z vzdrževanjem in menjavo žastarelihštrojnih virov.
Osnovne značilnosti oblaka so naslednje:

- razširljivost,
- dinamična konfiguracija storitev,
- dinamična namembnost in uporabnost,
- dosegljivost na zahtevo.

Poznamo tri tipe oblakov: javni oziroma komercialni oblak, ki je v lastni komercialnih ponudnikov, zasebni oblak, v lasti domače organizacije in hibridni oblak, ki združje prva dva. Modeli oblakov so trije, in sicer:

- programska oprema kot storitev (SaaS),
- platforma kot storitev (PaaS),
- infrastruktura kot storitev (IaaS).

V tem doktorskem delu se osredotočamo na tip infrastrukture kot storitve, saj je tu omogočen neposreden dostop do računskih virov. Uporabnik lahko izbere svoj operacijski sistem in namesti potrebno pogramsko opremo. V primerjavi z drugima tipoma oblaka, ta tip ponuja več fleksibilnosti, saj programska oprema ni vnaprej nameščena in lahko okolje

prilagodimo tako, da na njem izvajamo tudi eksperimente, ki za izdvedbo potrebujejo starejše različice programov ali knjižnic.

A.3.3 Primerjava grida in oblaka

Grid in oblak sta na videz podobni arhitekturi, a se med seboj precej razlikujeta. Obe arhitekturi sta sicer osnovani na porazdeljenih strojnih virih in sta razširljivi ter omogočata dostop do velikih računskih in podatkovnih kapacitet večji skupini uporabnikov [65]. Cilji njune uporabe so različni. Grid je namenjen računskim nalogam, ki ponavadi tečejo relativno kratek čas in vključujejo veliko vhodnih in izhodnih podatkov, oblak je namenjen dolgoročnemu gostovanju storitev [66]. Oblak sledi načelu enostavne uporabe, ki bi bila sprejemljiva in razumljiva za vse uporabnike, grid ima do svojih kapacitet omejen dostop in zagotovljeno varnost z uporabo delegiranih prijavnih podatkov in avtorizacije preko strežnika za upravljanje virtualnih organizacij VOMS. Oblak je enostavno razširljiv in ponuja na videz neomejeno število virov, medtem ko je grid omejen in pogosto ne zadosti vsem potrebam raziskovalcev. Tudi prilagajanje okolja za izvajanje je v gridu omejeno, v oblaku v celoti izvedljivo. Pri združevanju obeh arhitektur se je potrebno osredotočiti na prednosti obeh arhitektur in le-te uporabiti kot osnovo našega modela integracije.

A.4 Prednosti in izzivi združevanja arhitektur

Arhitektura grida ima nekaj težje premostljivih omejitev. Okolja za izvajanje nalog je mogoče le delno prilagoditi, saj je znotraj obstoječega operacijskega sistema mogoče urediti več različic različne programske opreme, ni pa mogoče uporabiti različnih operacijskih sistemov. Pogosto so omejitve povezane s strojno opremo, ki je v rabi (programska oprema prilagojena gonilnikom strojne opreme). Oblak, na drugi strani, ima preveč ohlapne mehanizme avtorizacije in overjanja ter nekoliko nižje zmogljivosti zaradi virtualizacije in sosednjih programov, ki tečejo na sistemu.
Združevanje arhitekture grida in oblaka ima naslednje prednosti:

- prenosljivost nalog iz grida v oblak,
- učinkovita izraba strojnih virov,
- razširljivost glede na potrebe,
- visoka zanesljivost in dosegljivost,

- napreden sistem za upravljanje uporabnikov in njihovih pravic izvajanja,
- zmanjšanje stroškov vzdrževanja,
- fleksibilnost pri dodeljevanju virov,
- izolacija virtualnih strežnikov od osnovnega operacijskega sistema,
- prilagodljiva okolja za izvajanje nalog.

Ker sta arhitekturi tako različni, jih ne moremo učinkovito in enostavno uporabiti za iste tipe nalog ali izvajanje iste programske opreme. Pri združevanju arhitektur naletimo na nekaj večjih izzivov [78]:

- nestandardizirani programski vmesniki za dostop do javnega oblaka,
- neenotni tipi slik sistemov,
- problem vključitve avtorizacije VOMS v javni oblak in avtentikacije preko digitalnih potrdil x509,
- upravljanje podatkov, njihova hramba in prenos,
- zagotavljanje ustreznega števila oblačnih instanc (kako zaznati večjo potrebo po virih in zagotoviti samodejni zagon instanc, koliko instanc zagnati, s kakšno programsko opremo, kdaj jih zaustaviti in izbrisati),
- zmanjšanje izvedbene zmogljivosti zaradi dodatne plasti v sistemu (virtualizacije).

A.5 Pristopi združevanja grida in oblaka in obstoječe rešitve

Pristopi k združevanju grida in oblaka se močno razlikujejo, saj pogramske rešitve, ki so na voljo, rešujejo specifične probleme raziskovalnih skupin in interpretirajo problematiko glede na študijo primera. Rešitve, ki so na voljo, običajno ne moremo prenesti v drugo okolje ali v drug javni oblak. Globalne programske rešitve ni.

Za učinkovito združevanje dveh različnih arhitektur je potrebno uporabiti sprejete standarde. Standardi namreč določajo protokole, specifikacije, možno uporabo tehnologije itn. Tehnologija grid je rešila že veliko problemov razpršenega računalništva, dobre

prakse je zato smiselno prenesti tudi v model oblaka. Govorimo o prenosu podatkov, pošiljanju nalog, razvrščanju nalog, overjanju, varnostnih mehanizmih in uporabniških vmesnikih. Največji izziv predstavlja zagotovo ogromna količina podatkov [78], saj so shrambe podatkov ponavadi entitete, ki so zunaj oblaka in zahtevajo prenos podatkov iz zunanjih virov. Zastavlja se vprašanje, kje podatke hraniti, kako jih prenesti v in iz oblaka. V javnih oblakih se obračunava tako diskovne kapacitete, kot tudi prenose. Oblačne instance, ki zagotavljajo boljše mrežne povezave, so tudi dražje. Dodaten izziv pri integraciji je implementacija mehanizmov za overjanje in avtorizacijo v oblaku. V javnem oblaku je ustvarjanje instanc omogočeno z uporabo uporabniškega imena in gesla, dostop do njih pa preko ssh-ključa. Dostop je enostaven, a ne zagotavlja visokega nivoja varnosti. V sklopu varnosti naletimo še na dodatno prepreko. Za vse ustvarjene instance je potrebno določiti, v katero varnostno skupino sodijo. Varnostna skupina vsebuje nabor pravil, preko katerih vrat bo omogočen dostop do virtualnih strežnikov. V izogib težavam pri dostopu, uporabniki navadno odprejo kar vsa vrata, kar je iz varnostnega vidika sporno. V primeru rabe odprtega dostopa na ravni hipernadzornika, je potrebno požarni zid nastaviti znotraj virtualnega strežnika. Dostop mora biti namreč omejen in pod nadzorom.

Trenutno se arhitektura grid ne zaveda nobenih oblačnih virov. Oblačni viri morajo biti registrirani v informacijskih storitvah, tako lahko odjemalci komunicirajo z njimi in jih uporabimo za izvajanje računskih nalog.

A.6 ARC-CC: združevanje grida in oblaka s pomočjo vmesne programske opreme ARC

Preizkušali smo modele združevanja grida in oblaka na način, da bi obstoječa infrastruktura grid omogočala tako izvedbo na fizičnih kot virtualnih virih, a smo sklenili, da je virtualizacija gruče grid z vmesno programsko opremo ARC bolj ustrezna, saj omogoča uporabo vseh javnih oblakov in olajša implementacijo številnih komponent samega sistema. Z uporabo predloge, narejene v programu Packer [136] in uporabo skript, ki se izvedejo po zagonu sistema, smo vzpostavili virtualno gručo grid, ki uporablja vmesno programsko opremo ARC. V predlogo lahko vključimo katerokoli sliko sistema (navadno so slike sistema že na voljo v imeniku ponudnika storitev v oblaku, lahko pa v imenik dodamo tudi svoje slike sistemov). Najprej postavimo glavno vozlišče, t.j. računsko vozlišče ARC-CE. Na njem tečejo osnovne storitve grid: upravljavec grid, informacijske storitve, strežnik gridftpd, začasna hramba podatkov in upravljavec gruče HTCondor [16].

Delovna vozlišča se priključujejo gruči samodejno, s pomogočjo programa Consul [144]. V datoteko *hosts* so dodani IP-naslovi strežnikov, prav tako v datoteko *exports*, ki služi nastavitvam storitve NFS. Vsa delovna vozlišča, na katerih se izvajajo naloge, vsebujejo konfiguracijsko datoteko za upravljavec gruče. Le-ta vsebuje podatke o glavnem vozlišču, upravljavcu in nastavitvah okolja za izvajanje. Za varnost lahko uporabimo lastno podpisano digitalno potrdilo, ali pa na gručo prenesemo digitalno potrdilo, podpisano iz strani EUgridPMA [131]. Če se odločimo za slednje, lahko na gruči omogočimo različne virtualne organizacije, ki določajo tudi pravila uporabe gruče. Uporabniki naloge pošiljajo na gručo na enak način kot na običajno gručo grid. Registracija nalog grid in storitev grid se izvaja samodejno. Zaradi uspešne implementacije informacijskih storitev, lahko uporabnik nemoteno spremlja stanje njegovih nalog. Uporabo okolij za izvajanje (RTE) smo vključili že v samo sliko sistema, določeno programsko opremo smo namestili z uporabo skript, ki se izvedejo po samem zagonu virtualnih strežnikov. Za avtorizacijo in overjanje smo uporabili enak pristop, kot ga uporabljamo na gručah grid, in sicer sistem VOMS in delegirane prijavne podatke.

A.7 Rezultati uporabe združjive arhitekture

Združljiv model ARC-CC smo preizkusili v zasebnem oblaku OpenStack in v javnem oblaku Amazon EC2. Izvedbo smo primerjali s tisto na običajni gruči grid, ki sestoji iz fizičnih strežnikov. Najprej smo na gruči izvedli primerjalne preizkuse CPU-ja in RAM-a, nato test arhiviranja in kasneje simulacije z uporabo programske opreme NAMD, ki je zelo priljubljena med našimi uporabniki, ki opravljajo raziskave na področju kemije, biokemije in biologije. Rezultati na gruči ARC-CC so bili primerljivi na zasebnem in javnem oblaku, a slabši kot na običajni gruči. Izvedba je bila na virtualizirani gruči za 15% slabša, kar je v veliki meri tudi posledica virtualizacije same. Osnovni testi so pokazali, da izvedbenih izgub na pomnilniku praktično ni zaznati, procesorskih izgub je okoli 7-8%Ŗazširitev simulacij na več procesorjev se je izkazala za učinkovito v vseh okoljih. Ob tem velja opozoriti, da izvedba zavisi tudi od tipa instance, ki jo izberemo na javnem oblaku in od ostalih virtualnih strežnikov, ki tečejo na isti strojni opremi. T.i. instance za splošno rabo dosegajo bistveno slabše rezultate kot računske instance, razlika je več kot 30%Ìnstance za splošno rabo so namreč prilagojene gostovanju nezahtevnih storitev, kot je gostovanje baze, spletnega strežnika itn. Te instance za računsko znanstveno obdelavo niso primerne. Računske instance sicer dosegajo nekoliko slabše rezultate kot fizična vozlišča, a je njihova velika prednost ta, da uporabnik pridobi dostop do neobdelanih virov,

ki jih lahko prilagaja, glede na potrebe njegovega eksperimenta. V gridu so te prilagoditve mogoče le v omejenem obsegu.

A.8 Zaključek

Tehnično združevanje zasebnih in javnih oblakov z arhitekturo grid, je trenutno ena glavnih tem večih raziskovalnih ustanov in projektov. V tem doktorskem delu smo predstavili prednosti, ki jih nova združljiva in porazdeljena arhitektura prinaša: enostavna migracija aplikacij iz ene arhitekture v drugo, zmanjšanje operativnih stroškov, boljša izraba strojnih virov, prilagodljiva okolja za izvajanje nalog, elastičnost in razširljivost infrastrukture.

Predstavili smo obstoječe poskuse integracij grida in oblaka. Pristopi k združevanju so različni, praviloma so rešitve prilagojene posamezni raziskovalni disciplini ali specifični postavitvi in so težje prenosljive v drugo okolje. Naša implementacija združitve grida in oblaka omogoča postavitev gruče grid v oblaku, s pomočjo vmesne programske opreme ARC, kar ustreza analogiji razširitve oblaka (ang. cloud bursting). V modelu ARC-CC je mogoče izvajati tako naloge grid, kot tudi serijske naloge, ki se pošiljajo neposredno na upravljavca gruče. Naša hibridna arhitektura omogoča enostavno uporabo virov v oblaku takrat, ko so kapacitete v gridu polne ali nedosegljive. Uporabniku za izvajanje nalog v oblaku ni potrebno spreminjati kode, zagonskih programov ali delovnega toka. Rešitev je uporabniku prijazna, saj je okolje grid v oblaku vzpostavljeno v roku nekaj minut in pošiljanje nalog v navidezno gručo poteka povsem transparentno. ARC-CC lahko postavimo v kateremkoli javnem ali zasebnem oblaku, lahko ga vključimo tudi v našo obstoječo infrastrukturo na Arnesu in ga uporabimo v vseh raziskovalnih disciplinah, saj je potrebno prilagoditi samo programsko opremo, ki se bo na virtualni gruči grid namestila. Dostop do ARC-CC je omogočen preko odjemalca ARC ali DIRAC. Okolja za izvajanje nalog so prevedena v slike sistema in vsebujejo vso programsko opremo in nastavitve, ki jih za delo potrebuje določena raziskovalna skupina. Ker ponudniki javnih oblakov največkrat ponujajo že izdelane slike slistemov, z minimalno namestitvijo programske opreme, smo sisteme ponastavljali tudi s skriptami, ki so se na virtualnih strežnikih izvajale po uspešno vzpostavljenem sistemu. Zagon teh skript je vključen v samo predlogo, ki je ustvarjena s programom Packer.

Gruča grid, z vmesno programsko opremo ARC, je nameščena z uporabo konfiguracijske datoteke in programom Packer. Rešitev lahko prenesemo v različne zasebne in javne oblake. Vzpostavitev potrebnih storitev in samodejno dodajanje vozlišč v gručo

smo omogočili z uporabo programa Consul, kompleksno upravljanje uporabnikov smo iz grida v celoti preneseli v oblak. Problem ostajajo visokoprepustne naloge, ki zahtevajo veliko I/O-operacij in veliko diskovnega prostora. V tej fazi smo se zato raje osredotočili na visokozmogljive naloge, ki zahtevajo veliko procesorske moči, a nimajo veliko vhodnih in izhodnih podatkov. Analizo zmogljivosti rešitve ARC-CC smo preverjali na zasebnem oblaku OpenStack in v javnem oblaku Amazon EC2. Zagnali smo simulacije, ki so procesorsko, ne pa tudi diskovno zahtevne, saj smo se ob preverjanju delovanja koncepta želeli izogniti težavam z diskovno hrambo in prenosi. Rezultati na zasebnem in javnem oblaku so bili podobni. Gručo ARC smo vzpostavili v nekaj minutah, pri zagonu simulacij smo beležili le 1% izgub zmogljivosti pomnilnika in okoli 7% izgub procesorske moči. Kljub vsemu velja omeniti, da rezultati testov niso bili vedno enaki, razlike so bile tudi do 15% kar je mogoče pripisati ostalim virtualnim strežnikom, ki so gostovali na isti strojni opremi. Omenjene razlike otežujejo realno oceno, koliko časa bo posamezna naloga potrebovala za izvršitev in temu primerne težave s pripravo opisa naloge, kjer zahtevamo ustrezne količine virov. Da bi preprečili nasilno prekinitev nalog, raje zahtevamo nekoliko več računskih virov, s tem onemogočimo optimalno izkoriščenost strojnih virov, ki jo sicer virtualizacija omogoča. V zasebnem oblaku lahko sami vplivamo na to, katere virtualne strežnike bomo gostili na istem fizičnem strežniku in s tem omenjene težave izničimo ali vsaj omilimo, medtem ko v javnem oblaku te možnosti nimamo. Težave se lahko pojavijo tudi z zakasnitvami. Preizkušali smo nekaj možnih postavitev, na podlagi katerih lahko zaključimo, da je bistvena izbira tipa diska. Če smo uporabili EBS hrambo Amazona, je bila izvedba slabša, kot pri uporabi diskov SSD. Zakasnitve lahko ublažimo tudi z uporabo predpomnilnika CPU in RAM, pisanjem neposredno v RAM, šele nato na disk, arhiviranjem podatkov itn.

Na virtualni gruči ARC-CC smo zagnali biokemijske analize z uporabo programa NAMD. Rezultati kažejo, da je razširljivost v vseh okoljih učinkovita in, da več procesorjev za simulacijo pomeni njeno hitrejšo izvršitev. Končna zmogljivost je bila slabša kot na fizičnih strežnikih. Če bi za analizo na običajni gruči grid potrebovali 12 dni, bi na virtualni gruči potrebovali 14 dni. Izguba zmogljivosti je skoraj 15%.Take rezultate smo pričakovali. Ne gre namreč za slabše delovanje virtualne gruče same, temveč za delovanje povezano z virualizacijo. Z vpeljavo hipernadzornika, vpeljemo v sistem dodatni nivo, ki zahteva vire. Poleg tega ni enak izkoristek predpomnilika za CPU in RAM, do diska dostopa več aplikacij hkrati, druge aplikacije lahko povzročajo prekinitve CPU-ja itn. Izvedba programa je odvisna tudi od ostalih virtualnih strežnikov, ki delijo iste strojne vire.

reqib zagotavlja visok nivo zasebnosti in varnosti.

A.9 Nadaljni razvoj rešitve ARC-CC

Naš model združevanja grida in oblaka je pomemben korak naprej v rabi porazdeljene oblačne infrastrukture kot dodatnih virov za znanstveno obdelavo podatkov, ko so obstoječe kapacitete polne. Arhitekturo lahko nadgradimo z namestitvijo rešitve ARC Rainbow [99], ki omogoča izvajanje nalog na operacijskem sistemu Windows. Hipernadzornik KVM je v zasebnih oblakih smiselno zamenjati z uporabo LXC, saj je izvedba primerljiva tisti na fizičnih strežnikih. V rešitev je potrebno vključiti tudi neposredno in samodejno dodeljevanje oblačnih instanc glede na smiselnost izvedbe nalog v oblaku in glede na zasedenost gruče, kar bi bilo mogoče rešiti s prilagoditvami programa Cloudscheduler [108], Vcycle [140] ali celo s popravki storitve *condor rooster* [154]. Storitev *condor_rooster* je sicer namenjena zagonu vozlišč, ki so v stanju mirovanja, a bi jo bilo mogoče spremeniti za to, da lahko obstoječi infrastrukturi dodaja oblačne vire, ko so ti potrebni.

Nekaj izzivov nas še čaka. Uporaba rešitve ARC-CC za podatkovno zahtevne naloge, ki zajema tudi iskanje rešitev za slabe ali drage omrežne povezave ter za prenose vhodnih in izhodnih podatkov. Vsekakor pa je razvoj v prihodnosti odvisen od uporabnikov, njihovih potreb in delovnih procesov.

BIBLIOGRAPHY

[1] Uwe Schwiegelshohn, Rosa M. Badia, Marian Bubak, Marco Danelutto, Schahram Dustdar, Fabrizio Gagliardi, Alfred Geiger, Ladislav Hluchy, Dieter Kranzlmüller, Erwin Laure, Thierry Priol, Alexander Reinefeld, Michael Resch, Andreas Reuter, Otto Rienhoff, Thomas Rüter, Peter Sloot, Domenico Talia, Klaus Ullmann, Ramin Yahyapour, and Gabriele von Voigt. Perspectives on grid computing. *Future Generation Computer Systems*, 26(8):1104–1115, October 2010. ISSN 0167739X. doi: 10.1016/j.future.2010.05.010.

[2] Maozhen Li and Mark Baker. *The grid: core technologies*. Wiley, Hoboken, NJ, 2005. ISBN 0470094176 9780470094174.

[3] Borko Furht and Armando Escalante, editors. *Handbook of Cloud Computing*. Springer US, Boston, MA, 2010. ISBN 978-1-4419-6523-3, 978-1-4419-6524-0.

[4] Anand Ranganathan and Roy H. Campbell. What is the complexity of a distributed computing system? *Complexity*, 12(6):37–45, July 2007. ISSN 10762787, 10990526. doi: 10.1002/cplx.20189.

[5] Salim Hariri and Anljjan Varma. High-performance distributed computing: Promises and challenges. *Concurrency: Practice and Experience*, 5(4):233–238, 1993.

[6] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Cloud computing and grid computing 360-degree comparedGrid Computing Environments Workshop, 2008. GCE'08*, pages 1–10, 2008.

[7] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, Á. Frohner, and A. Gianoli. VOMS, an Authorization System for Virtual Organizations. In *VOMS, an Authorization System for Virtual Organizations*, volume 2970, pages 33–40. Springer Berlin Heidelberg, 2004.

[8] Gabriel Mateescu, Wolfgang Gentzsch, and Calvin J. Ribbens. Hybrid Computing—Where HPC meets grid and Cloud Computing. *Future Generation Computer Systems*, 27(5):440–453, May 2011. ISSN 0167739X. doi: 10.1016/j.future.2010.11.003.

[9] Adam A. Hunter, Andrew B. Macgregor, Tamas O. Szabo, Crispin A. Wellington, and Matthew I. Bellgard. Yabi: An online research environment for grid, high performance and cloud computing. *Source code for biology and medicine*, 7(1):1, 2012.

[10] Miron Livny, Jim Basney, Rajesh Raman, and Todd Tannenbaum. Mechanisms for high throughput computing. *SPEEDUP journal*, 11(1):36–40, 1997.

[11] Ruben S. Montero, Rafael Moreno-Vozmediano, and Ignacio M. Llorente. An elasticity model for High Throughput Computing clusters. *Journal of Parallel and Distributed Computing, Special issue on cloud computing*, 71(6):750–757, June 2011.

[12] Ivona Brandic and Rajkumar Buyya. Special section: Recent advances in utility and cloud computing. *Future Generation Computer Systems*, 28(1):36–38, January 2012. ISSN 0167739X. doi: 10.1016/j.future.2011.06.001.

[13] José C. Cunha and Pedro D. Medeiros. Special Issue: Parallel and Distributed Computing (EuroPar 2005). *Concurrency and Computation: Practice and Experience*, 19(17):2183–2184, December 2007. ISSN 15320626, 15320634. doi: 10.1002/cpe.1267.

[14] OpenStack Operations Guide. OpenStack Cloud sofware - OpenStack operations guide, July 2014. URL `http://docs.openstack.org/ops-guide`.

[15] Lopez Garcia, A., Fernandez-del-Castillo, E., and Puel, M. Voms-aware identity service for openstack. In *EGI Community Forum*, 2013.

[16] Htcondor batch system. URL `https://research.cs.wisc.edu/htcondor/`.

[17] Amazon Elastic Compute Cloud. Amazon Elastic Compute Cloud, February 2013. URL `http://aws.amazon.com/ec2/instance-types/`.

[18] René Brun, Federico Carminati, and Giuliana Galli Carminati, editors. *From the Web to the Grid and Beyond*. The Frontiers Collection. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-23156-8, 978-3-642-23157-5.

[19] the Atlas Collaboration. The ATLAS Experiment at the CERN Large Hadron Collider. *Journal of Instrumentation*, 3(08):S08003, 2008.

[20] R. Brun, F. Carminati, and G. Galli Carminati. *From the web to the grid and beyond: Computing Paradigms Driven by High-Energy Physics*. The Frontiers Collection. Springer, 2012.

[21] Pascale Vicat-Blanc. *Computing networks: from cluster to cloud computing*. ISTE ; Wiley, London; Hoboken, N.J., 2011. ISBN 9781848212862 1848212860.

[22] Naidila Sadashiv and SM Dilip Kumar. Cluster, grid and cloud computing: A detailed comparison. In *Computer Science & Education (ICCSE), 2011 6th International Conference on*, pages 477–482. IEEE, 2011.

[23] R. P. Bruin, T. O. H. White, A. M. Walker, K. F. Austen, M. T. Dove, R. P. Tyer, P. A. Couch, I. T. Todorov, and M. O. Blanchard. Job submission to grid computing environments. *Concurrency and Computation: Practice and Experience*, 20(11):1329–1340, August 2008. ISSN 15320626, 15320634. doi: 10.1002/cpe.1290.

[24] Frederic Magoules, Jie Pan, Kiat-An Tan, and Abhinit Kumar. *Introduction to Grid Computing*. CRC press, 2009.

[25] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: The Condor experience. *Concurrency-Practice and Experience*, 17(2-4): 323–356, 2005.

[26] European Middleware Initiative (EMI) Execution Service, 2011. URL https://twiki.cern.ch/twiki/bin/view/EMI/EmiExecutionService.

[27] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, June 2009. ISSN 0167739X. doi: 10.1016/j.future.2008.12.001.

[28] Peter deutsch: 8 fallacies about distributed computing. URL https://blogs.oracle.com/jag/resource/Fallacies.html.

[29] Arnon Rotem-Gal-Oz. Fallacies of distributed computing explained. *RGOArchitects*, 2006.

[30] Brendan Gregg. *System Performance: enterprise and the Cloud*. Edwards Brothers, 2013.

[31] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Elsevier inc., 2004.

[32] Ian Foster. What is grid? A Three Point Checklist, 2002.

[33] P. Bunčič and F. Carminati. A discussion on virtualization in grid computing. *The Frontiers Collection, Springer*, From the Web to the Grid and Beyond: 155–175, 2012.

[34] B. Sotomayor, S. Ruben Montero, I. M. Lorente, and I. Foster. Virtual Infrastructure Management in Private and Hybrid Clouds. *Internet Computing, IEEE*, 13(5): 14–22, 2009.

[35] Ian Foster and Carl Kesselman. Computational grids. In *Vector and Parallel Processing—VECPAR 2000*, pages 3–37. Springer, 2001.

[36] Rajesh Kalmady, Digamber Sonvane, Phool Chand, Kislay Bhatt, Kumar Vaibhav, Piotr Nyczyk, and Zdenek Sekera. Monitoring the availability of grid services using SAM and gridview. In *Grid Computing*, pages 163–168. Springer, 2009.

[37] Ismael M. Carrion, Eduardo Huedo, and Ignacio M. Llorente. Interoperating grid infrastructures with gridway metascheduler. *Concurrenct and Computation: practice and Experience*, 2012. ISSN 15320626. doi: 10.1002/cpe.2971.

[38] Les Robertson. Computing Services for LHC: from Clusters to Grids. *The Frontiers Collection, Springer-Verlag Berlin Heidelberg*, From the Web to the Grid and Beyond:69–89, 2012.

[39] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid. *Berman et al.[2]*, pages 171–197, 2003.

[40] I. Foster, J. Geisler, W. Nickless, W. Smith, and S. Tuecke. Software Infrastructure for the I-WAY High Performance Distributed Computing Experiment. In *Proc. 5th IEEE Symposium on High Performance Distributed Computing*, pages 562–571, 1997.

[41] David Villegas, Ivan Rodero, Liana Fong, Norman Bobroff, Yanbin Liu, Manish Parashar, and S. Masoud Sadjadi. The Role of Grid Computing Technologies in Cloud Computing. In Borko Furht and Armando Escalante, editors, *Handbook of Cloud Computing*, pages 183–218. Springer US, Boston, MA, 2010. ISBN 978-1-4419-6523-3, 978-1-4419-6524-0.

[42] Egi-inspire website, 2016. URL https://www.egi.eu/about/egi-inspire.

[43] Nordugrid. Nordugrid - Grid Research and Development collaboration. URL www.nordugrid.org.

[44] Dietmar W. Erwin. UNICORE—a Grid computing environment. *Concurrency and Computation: Practice and Experience*, 14(13-15):1395–1410, 2002.

[45] Michal Novotnỳ. Job scheduling with the SLURM resource manager. 2012.

[46] Pbs batch system. URL http://www.adaptivecomputing.com/products/open-source/torque/.

[47] Open Grid Scheduler. URL http://gridscheduler.sourceforge.net.

[48] Usha Divakarla and Geetha Kumari. An overview of cloud computing in distributed systems. In *AIP Conference Proceedings*, volume 1324, pages 184–186, Chandigarh,India, December 2010. American Institute of Physics.

[49] L.M Vaqero et al. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2009.

[50] Keqin Li. Optimal Load Distribution for Multiple Heterogeneous Blade Servers in a Cloud Computing Environment. *Journal of Grid Computing*, 11 (1):27–46, March 2013. ISSN 1570-7873, 1572-9184. doi: 10.1007/s10723-012-9239-y.

[51] Lizhe Wang, Jie Tao, Marcel Kunze, Alvaro Canales Castellanos, David Kramer, and Wolfgang Karl. Scientific Cloud Computing: Early Definition and Experience. In *HPCC*, volume 8, pages 825–830, 2008.

[52] Massimo Cafaro and Giovanni Aloisio, editors. *Grids, Clouds and Virtualization*. Computer Communications and Networks. Springer London, London, 2011. ISBN 978-0-85729-048-9, 978-0-85729-049-6.

[53] Digitalocean. URL https://www.digitalocean.com/.

[54] Edward Walker. Benchmarking Amazon EC2 for high-performance scientific computing. *LOGIN*, 33(5): 18–23, 2008.

[55] Google compute engine. URL https://cloud.google.com/compute/.

[56] Rackspace, 2016. URL https://www.rackspace.com/.

[57] Philipp C. Heckel. Hybrid Clouds: Comparing Cloud Toolkits, 2010. URL http://www.philippheckel.com/files/hybrid-cloud-paper.pdf.

[58] OpenNebula. OpenNebula. URL http://opennebula.org/.

[59] Rick Bradshaw and Piotr T. Zbiegiel. Experiences with Eucalyptus: Deploying an Open Source Cloud. In *LISA'10 Proceedings of the 24th international conference on Large installation system administration*, pages 1–16. Usenix Association Berkeley, 2010.

[60] OpenStack. OpenStack - Open Source Cloud Computing Software. URL http://www.openstack.org/.

[61] oVirt virtualization platform, February 2013. URL http://www.ovirt.org/Home.

[62] L. Yuxi and W. Jianhua. Research on Comparison of Cloud Computing and Grid Computing. *Research Journal of Applied Sciences*, 4, 2012.

[63] HwaYoung Jeong and JongHyuk Park. An efficient cloud storage model for cloud computing environment. In *Advances in Grid and Pervasive Computing*, pages 370–376. Springer, 2012.

[64] Simon Ostermann, Radu Prodan, and Thomas Fahringer. Extending grids with cloud resource management for scientific computing. In *Grid Computing, 2009 10th IEEE/ACM International Conference on*, pages 42–49. IEEE, 2009.

[65] Thomas Rings, Geoff Caryer, Julian Gallop, Jens Grabowski, Tatiana Kovacikova, Stephan Schulz, and Ian Stokes-Rees. Grid and Cloud Computing: Opportunities for Integration with the Next Generation Network. *Journal of Grid Computing*, 7(3): 375–393, September 2009. ISSN 1570-7873, 1572-9184. doi: 10.1007/s10723-009-9132-5.

[66] Shantenu Jha, Andre Merzky, and Geoffrey Fox. Using clouds to provide grids with higher levels of abstraction and explicit support for usage modes. *Concurrency and Computation: Practice and Experience*, 21(8):1087–1108, 2009.

[67] Srikumar Venugopal, Krishna Nadiminti, Hussein Gibbins, and Rajkumar Buyya. Designing a resource broker for heterogeneous grids. *Software: Practice and Experience*, 38(8):793–825, 2008.

[68] Sergio Andreozzi, Massimo Sgaravatto, and Cristina Vistoli. Sharing a conceptual model of Grid resources and services. *arXiv preprint cs/0306111*, 2003. URL http://arxiv.org/abs/cs/0306111.

[69] R. Menday and Ph Wieder. GRIP: The Evolution of UNICORE towards a Service-Oriented Grid. In *Proc. of the 3rd Cracow Grid Workshop (CGW'03)*, pages 142–150, 2003. URL https://unicore.eu/documentation/files/menday-2003-GEU.pdf.

[70] Kate Keahey, Matei Ripeanu, and Karl Doering. Dynamic creation and management of runtime environments in the grid. In *Workshop on Designing and Building Web Services (to appear)*. Citeseer, 2003.

[71] Eduardo Huedo, Rafael Moreno-Vozmediano, Rubén S. Montero, and Ignacio M. Llorente. Architectures for Enhancing Grid Infrastructures with Cloud Computing. In Massimo Cafaro and Giovanni Aloisio, editors, *Grids, Clouds and Virtualization*, pages 55–69. Springer London, London, 2011. ISBN 978-0-85729-048-9, 978-0-85729-049-6. URL http://link.springer.com/10.1007/978-0-85729-049-6_3.

[72] P Nilsson, J Caballero, K De, T Maeno, A Stradling, T Wenaus, and the Atlas Collaboration. The ATLAS PanDA Pilot in Operation. *Journal of Physics: Conference Series*, 331(6):062040, December 2011. ISSN 1742-6596. doi: 10.1088/1742-6596/331/6/062040.

[73] Constantino Vázquez, Eduardo Huedo, Rubén S. Montero, and Ignacio M. Llorente. On the use of clouds for grid resource provisioning. *Future Generation Computer Systems*, 27(5):600–605, May 2011. ISSN 0167739X. doi: 10.1016/j.future.2010.10.003.

[74] V. Garonne, A. Yu Tsaregorodtsev, and I. Stokes-Rees. DIRAC. Technical Report CERN-LHCb-2004-090, CERN, Geneva, 2004.

[75] Georgiana Marin and others. Grid Computing Technology. *Database Systems Journal*, 2(3):13–22, 2011.

[76] Simon Ostermann, Radu Prodan, and Thomas Fahringer. Dynamic cloud provisioning for scientific grid workflows. In *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, pages 97–104. IEEE, 2010.

[77] Simon Ostermann, Alexandria Iosup, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. A performance analysis of EC2 cloud computing services for scientific computing. In *Cloud Computing*, pages 115–131. Springer, 2010.

[78] P. Mendez Lorenzo and Jamie Shiers. The Realities of Grid Computing. *The Frontiers Collection, Springer*, From the Web to the Grid and Beyond:91–114, 2012.

[79] Michal Procházka. PERUN - User and Resource Management System, February 2013. URL https://perun.cesnet.cz/web/media/REFEDS-2.6.2013.pdf.

[80] Alvaro Garcia. VOMS-aware identity service for OpenStack, 2013.

[81] A. Di Megliio, M. Reidel, Shahbaz Memon, Cal Loomis, and Davide Salomoni. Grids and Clouds Integration and Interoperability: an overview. In *International Symposium on Grids and Clouds and the Open Grid Forum (ISGC 2011 & OGF 31)*, page 112. Proceedings of Science, 2012. URL http://hal.in2p3.fr/in2p3-00836598/.

[82] SIENA European Roadmap on Grid and Cloud Standards for e-Science and Beyond. Technical report, Standards and Interoperability for eInfrastructure Implementation Initiative, May 2012.

[83] Gang Chen Yongjian Wang, Yaodong Cheng. *Production Grids in Asia*, chapter Interoperability between gLite and GOS, pages 155–173. Springer US, 2010.

[84] Constantino Vázquez Blanco, Eduardo Huedo, Rubén S. Montero, and Ignacio M. Llorente. Dynamic Provision of Computing Resources from Grid Infrastructures and Cloud Providers. In *IEEE 2009 Workshops at the Grid and Pervasive Computing*, pages 113–120. IEEE, May 2009. ISBN 978-0-7695-3677-4, 978-1-4244-4372-7. doi: 10.1109/GPC.2009.22.

[85] Piyush Harsh, Florian Dudouet, Roberto G. Cascella, Yvon Jégou, and Christine Morin. Using Open Standards for Interoperability-Issues, Solutions, and Challenges facing Cloud Computing. *6th International DMTF Academic Alliance Workshop on Systems and Virtualization Management: Standards and the Cloud*, 2012.

[86] Michael Hogan and Annie Sokol. NIST Cloud Computing Standards Roadmap. Technical report, National Institution of Standards and Technology, July 2013.

[87] Christine Morin, Yvon Jégou, Jérôme Gallard, and Pierre Riteau. Clouds: a new playground for the xtreemos grid operating system. *Parallel processing letters*, 19(03):435–449, 2009.

[88] Xtreemos - enabling linux for the grid. URL http://www.xtreemos.eu/.

[89] Matthias Hovestadt, Odej Kao, Andreas Kliem, and Daniel Warneke. Adaptive Online Compression in Clouds—Making Informed Decisions in Virtual Machine Environments. *Journal of Grid Computing*, 11 (2):167–186, June 2013. ISSN 1570-7873, 1572-9184. doi: 10.1007/s10723-013-9249-4.

[90] Ostermann, Simon, Prodan, Radu, and Fahringer, Thomas. Resource management for hybrid grid and cloud computing. In *Cloud Computing*, pages 179–194. Springer, 2010.

[91] European Grid Initiative. European Grid Initiative. URL http://www.egi.eu.

[92] María Arsuaga-Ríos, Seppo S Heikkilä, Dirk Duellmann, René Meusel, Jakob Blomer, and Ben Couturier. Using S3 cloud storage with ROOT and CvmFS. *Journal of Physics: Conference Series*, 664(2): 022001, December 2015. ISSN 1742-6588, 1742-6596. doi: 10.1088/1742-6596/664/2/022001.

[93] Elasticluster. Elasticluster. URL http://gc3-uzh-ch.github.io/elasticluster/.

[94] StarCluster. StarCluster. URL http://star.mit.edu/cluster/.

[95] Kate Keahey, Renato Figueiredo, Jos Fortes, Tim Freeman, and Maurcio Tsugawa. Science clouds: Early experiences in cloud computing for scientific applications. *Cloud computing and applications*, 2008, 2008.

[96] Scms.pro opensource supercomputer management system, 2016. URL http://scms.icybcluster.org.ua/.

[97] Davide Salomoni, Alessandro Italiano, and Elisabetta Ronchieri. WNoDeS, a tool for integrated Grid and Cloud access and computing farm virtualization. *Journal of Physics: Conference Series*, 331(5):052017, December 2011. ISSN 1742-6596. doi: 10.1088/1742-6596/331/5/052017.

[98] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software-Practice and Experience*, 32(2):135–64, 2002.

[99] Andrii Salnikov. RAINBOW - ARC in the cloud. In *RAINBOW - ARC in the cloud*, Helsinki, Finland, 2014.

[100] Larry Pezzaglia. CHOS in Production, Multiple Linux Environments on PDSF at NERSC, April 2012.

[101] Andrei Tsaregorodtsev. DIRAC services. In *EGI Community Forum Bari*, 2015.

[102] Sangmi Lee Pallickara, Marlon Pierce, Qunfeng Dong, and ChinHua Kong. Enabling large scale scientific computations for expressed sequence tag sequencing over grid and cloud computing clusters. *PPAM 2009 EIGHTH INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING AND APPLIED MATHEMATICS Wroclaw, Poland*, 2009.

[103] Franck Cappello, Samir Djilali, Gilles Fedak, Thomas Herault, Frédéric Magniette, Vincent Néri, and Oleg Lodygensky. Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid. *Future Generation Computer Systems*, 21(3):417–437, 2005.

[104] Gilles Fedak. XtremWeb: a generic global computing system. In *Cluster Computing and the Grid*, pages 582–587, Brisbane, May 2001. ISBN 0-7695-1010-8. doi: 10.1109/CCGRID.2001.923246.

[105] VENUS. VENUS-C. URL www.venus-c.eu.

[106] Wei Chen, Kuo-Cheng Yin, Don-Lin Yang, and Ming-Chuan Hung. Data Migration from Grid to Cloud Computing. *Applied Mathematics and Information Sciences - NSP*, 7(1):399–406, 2013.

[107] Andrew McNab, Cinzia Luzzi, and Federico Stagni. LHCb experience running jobs in virtual machines. Technical report, 2015.

[108] Ian Gable. HEP cloud production using the CloudScheduler/HTCondor architecture. In *CHEP*, Okinawa, Japan, April 2015.

[109] Keith R. Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf, Harvey J. Wasserman, and Nicholas J. Wright. Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. pages 159–168. IEEE, November 2010. ISBN 978-1-4244-9405-7. doi: 10.1109/CloudCom.2010.69.

[110] Katarzyna Keahey, Mauricio Tsugawa, Andrea Matsunaga, and Jose Fortes. Sky computing. *Internet Computing, IEEE*, 13(5):43–51, 2009.

[111] Loomis C,, Airaj M., Begin M-E., Floros E,, Kenny S., and O'Callaghan D,. *European Research Activities in Cloud Computing: StratusLab Cloud Distribution*. Cambridge Scholars Publishing, 2012. ISBN 1-4438-3507-2.

[112] Ioannis Konstantinou, Evangelos Floros, and Nectarios Koziris. Public vs private cloud usage costs: the StratusLab case. In *CloudCP '12 Proceedings of the 2nd International Workshop on Cloud Computing Platforms*, New York, 2012. ACM. ISBN 978-1-4503-1161-8. doi: 10.1145/2168697.2168700.

[113] Xingchen Chu, Krishna Nadiminti, Chao Jin, Srikumar Venugopal, and Rajkumar Buyya. Aneka: Next-generation enterprise grid platform for e-science and e-business applications. In *e-Science and Grid Computing, IEEE International Conference on*, pages 151–159. IEEE, 2007.

[114] Beniamino Di Martino, Giuseppina Cretella, and Antonio Esposito. *Cloud Portability and Interoperability*. SpringerBriefs in Computer Science. Springer International Publishing, Cham, 2015. ISBN 978-3-319-13700-1, 978-3-319-13701-8.

[115] Helix nebula - the science cloud, 2016. URL http://www.helix-nebula.eu.

[116] Elasticluster documentation. Elasticluster documentation. URL http://elasticluster.readthedocs.org/en/latest/.

[117] Ansible. ANSIBLE. URL http://www.ansible.com/.

[118] Bright computing. URL http://www.brightcomputing.com.

[119] boto API. boto: A Python interface to Amazon Web Services. URL http://boto.cloudhackers.com/en/latest/.

[120] Arnes. Arnes. URL www.arnes.si/en.

[121] Thomas Rings and Jens Grabowski. Pragmatic Integration of Cloud and Grid Computing Infrastructures. pages 710–717. IEEE, June 2012. ISBN 978-1-4673-2892-0, 978-0-7695-4755-8. doi: 10.1109/CLOUD.2012.77.

[122] Synnefo. Synnefo. URL https://www.synnefo.org/.

[123] Cloudstack. Cloudstack. URL https://cloudstack.apache.org.

[124] CentOS Project. CentOS Project. URL https://www.centos.org.

[125] G Behrmann, P Fuhrmann, M Grønager, and J Kleist. A distributed storage system with dCache. *Journal of Physics: Conference Series*, 119(6):062014, July 2008. ISSN 1742-6596. doi: 10.1088/1742-6596/119/6/062014.

[126] Michal Novotný. Job scheduling with the SLURM resource manager. Master's thesis, Masaryk University, Brno, 2009.

[127] Andy B. Yoo, Morris A. Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing*, pages 44–60. Springer, 2003.

[128] Brett Bode, David M. Halstead, Ricky Kendall, Zhou Lei, and David Jackson. The portable batch scheduler and the maui scheduler on linux clusters. In *Usenix, 4th Annual Linux Showcase & Conference*, 2000.

[129] Gentzsch, W. Sun Grid Engine: towards creating a compute power grid. In *Proceedings First IEEE/ACM International Symposium on*, pages 35 – 36, Brisbane, 2001. ISBN 0-7695-1010-8. doi: 10.1109/CCGRID.2001.923173.

[130] Leszek Sliwko and Vladimir Getov. Workload Schedulers-Genesis, Algorithms and Comparisons. *International Journal of Computer Science and Software Engineering*, 4(6):141–155, 2015.

[131] EUgridPMA. EUgridPMA. URL https://www.eugridpma.org/.

[132] Lee, C.A. and Desai, N. Approaches for Virtual Organization Support in OpenStack. In *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, pages 432 – 438, Boston, MA, March 2014. doi: 10.1109/IC2E.2014.35.

[133] XRSL. XRSL - Extended Resource Specification Language. URL http://www.nordugrid.org/documents/xrsl.pdf.

[134] Massimo Sgaravatto. Specification of the JDL attributes supported by the CREAM CE service, 2011.

[135] Russell Lock and Ian Sommerville. Grid Security and its use of X. 509 Certificates. *Department of Computer Science Lancaster University. Funded by EPSRC project studentship associated with the UK EPSRC DIRC project grant GR N*, 13999, 2002.

[136] Packer. Packer. URL https://packer.io/.

[137] Simon Ostermann, Kassian Plankensteiner, and Radu Prodan. Using a new event-based simulation framework for investigating resource provisioning in Clouds. *Scientific Programming*, 19(2):161–178, 2011.

[138] Terraform - collaborative infrastructure automation, 2016. URL https://www.terraform.io/.

[139] Apache libcloud, 2016. URL https://libcloud.apache.org/.

[140] Vcycle: Vm lifecycle management, 2016. URL https://www.gridpp.ac.uk/vcycle.

[141] Mike Burrows. The Chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 335–350. USENIX Association, 2006.

[142] Zookeeper. Zookeeper. URL https://zookeeper.apache.org/.

[143] CoreOS/etcd. CoreOS/etcd. URL https://github.com/coreos/etcd.

[144] Consul. Consul. URL https://www.consul.io/.

[145] David W. Chadwick, Kristy Siu, Craig Lee, Yann Fouillat, and Damien Germonville. Adding Federated Identity Management to OpenStack. *Journal of Grid Computing*, 12(1):3–27, March 2014. ISSN 1570-7873, 1572-9184. doi: 10.1007/s10723-013-9283-2.

[146] Lopez Garcia, A., Fernandez-del-Castillo, E., and Puel, M. Identity Federation with VOMS in Cloud Infrastructures. *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on on Cloud Computing Technology and Science*, 1: 42–48, 2013.

[147] Amazon instance types, 2016. URL https://aws.amazon.com/ec2/instance-types/.

[148] sysbench. Sysbench benchmarking tool. URL https://launchpad.net/sysbench.

[149] Phoronix. Phoronix Test Suite. URL http://www.phoronix-test-suite.com/.

[150] Adam K.L. Wong and Andrzej M. Goscinski. A VMD Plugin for NAMD Simulations on Amazon EC2. *Procedia Computer Science*, 9:136–145, 2012. ISSN 18770509. doi: 10.1016/j.procs.2012.04.015.

[151] James C. Phillips, Gengbin Zheng, and Sameer et al. Kumar. NAMD: Biomolecular simulation on thousands of processors. In *Supercomputing, ACM/IEEE 2002 Conference*, pages 36–36. IEEE, 2002.

[152] Apoa1. Apoa1 NAMD benchmark. URL http://www.ks.uiuc.edu/Research/namd/utilities.

[153] NAMD. NAMD - Scalable molecular dynamics. URL http://www.ks.uiuc.edu/Research/namd/.

[154] Derek Weitzel. *Campus Grids: A framework to facilitate resource sharing*. PhD thesis, University of Nebraska, 2011. URL http://osg-docdb.opensciencegrid.org/0010/001052/001/DerekWeitzelThesis.pdf.